DRAFT Test Teach

DS2 Programming: Essentials

Course Notes

DS2 Programming: Essentials Course Notes was developed by David Ghan and Mark Jordan. Additional contributions were made by Rob McAfee. Editing and production support was provided by the Curriculum Development and Support Department.

DS2 Programming: Essentials Course Notes

Book code E70481, course code LWDS2E/, prepared date 02Sep2015. LWDS2E_003

ISBN 978-1-62960-055-0

Table of Contents

Course Description

This course focuses on DS2, which is a fourth-generation SAS proprietary language for advanced data manipulation. DS2 enables parallel processing and storage of large data with re-usable methods and packages. This course is designed for programmers with large data who want to use modern programming techniques and structures.

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at http://support.sas.com/training/ as well as in the Training Course Catalog.

For a list of other SAS books that relate to the topics covered in this course notes, USA customers can contact the SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the SAS Bookstore on the web at http://support.sas.com/publishing/ for a complete list of books and a convenient order form.

Prerequisites

This course is not appropriate for beginning SAS software users. Before attending this course, you should have several months of SAS programming experience, or have taken the SAS® Programming 2: Data Manipulation Techniques course. You should also have a solid background in ANSI SQL: 1999.

Chapter 1 Introduction

1.1 Introduction to DS2

Objectives

- Describe the DS2 programming language.
- Compare the DS2 DATA program to Base SAS DATA step execution.
- Identify when it is most appropriate to use the DS2 language.

3

What Is DS2?

DS2 is a new SAS programming language.

- It is included with Base SAS.
- It has syntax similar to the Base SAS DATA step.
- It provides advanced data manipulation techniques.

Base SAS DATA Step

```
data _null_;
   Text='Hello, World!';
   put Text=;
run;
```

DS2 DATA program

```
proc ds2;
data _null_;
   method init();
      Text='Hello, World!';
      put Text=;
   end;
enddata;
run;
quit;
```

4

1.01 Multiple Answer Poll

Which of these SAS programming languages have you used?

a. Base SAS
b. SQL
c. DS2
d. macro
e. other

5

What Is New in DS2?

DS2 natively supports ANSI SQL data types for precise data manipulation.

Examples:

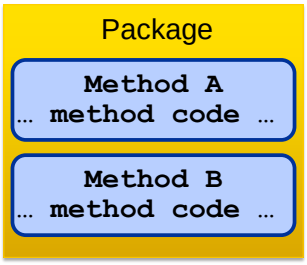
Data Type	Examples
Fractional Numerics	DECIMAL, DOUBLE, FLOAT, REAL
Integer Numerics	BIGINT, INTEGER, SMALLINT, TINYINT
Date and Time	DATE, TIME, TIMESTAMP
Character	CHAR, NCHAR, VARCHAR, NVARCHAR

SAS numeric variables are processed as DOUBLE.
SAS character variables are processed as CHAR.

6

What Is New in DS2?

DS2 improves the extensibility and reusability of code through the use of methods and packages.



Methods and packages can be pre-defined or user-defined.

7

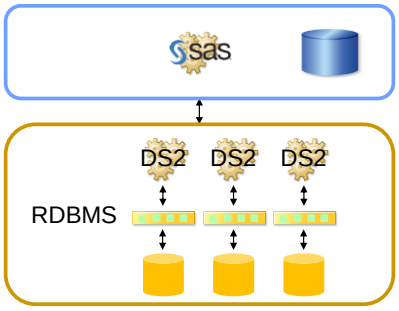
What Is New in DS2?

- The Base SAS DATA step processes each observation sequentially.
- DS2 can process observations in parallel.

8

What Is New in DS2?

With the SAS In-Database Code Accelerator, DS2 threads can be executed in parallel in-database.



9

How Are DS2 Programs Created?

DS2 programs are written using the following Base SAS procedures:

- PROC DS2
- PROC HPDS2

10

Where Can DS2 Programs Execute?

PROC DS2 can execute DS2 programs in the following:

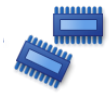
- Base SAS
- a supported database in which the SAS In-Database Code Accelerator is installed

11

Where Can DS2 Programs Execute?

PROC HPDS2 can execute DS2 programs in the following modes:

- single machine – threaded processing on the SAS client machine
- distributed – threaded processing using SAS High-Performance Analytics Server infrastructure
  - client-data mode
  - alongside the database
  - alongside HDFS (Hadoop)
  - alongside LASR



12

When to Use DS2?

DS2 can access data from the following data sources:

- Aster
- DB2 (UNIX and PC)
- Pivotal Greenplum
- Hadoop (Hive and HDMD)
- Memory Data Store (MDS)
- MYSQL
- Netezza
- ODBC-compliant databases
- Oracle
- PostgreSQL
- SAP (read only)
- SAP HANA
- SAS data sets
- SASHDAT files
- SPD Engine data sets
- Sybase IQ
- Teradata

13

The most current list of supported data sources can be found online in the *SAS® 9.4 DS2 Language Reference*.

When to Use DS2?

DS2 programs are best suited for applications that do the following:

- take advantage of threaded processing
  - are computationally complex
  - can execute in massively parallel processing (MPP) databases
- require the precision that the DS2 data types offer
- leverage the convenient reusability of DS2 methods and packages

14

1.2 Course Logistics

Objectives

- Describe the data that is used in the course.
- Designate the editors and processing mode that are available for workshops.
- Specify the naming convention that is used for the course files.
- Define the three levels of exercises.
- Navigate the Help facility.

17

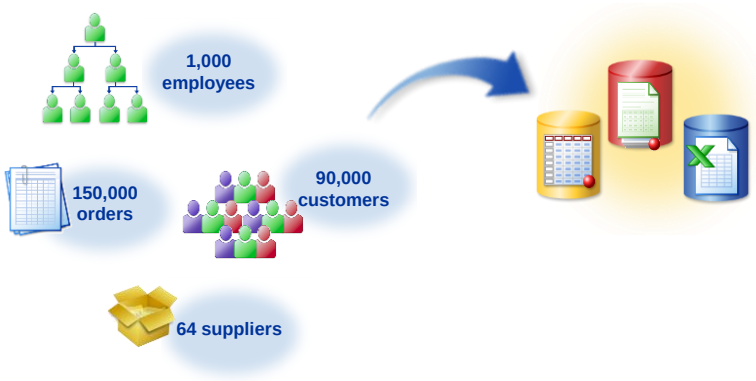
Orion Star Sports & Outdoors

This course focuses on a fictitious global sports and outdoors retailer that has traditional stores, an online store, and a large catalog business.

18

Orion Star Data

Large amounts of data are stored in transactional systems in various formats.



19

SAS Programming Interfaces

In this course, three interfaces are available for you to use during workshops.

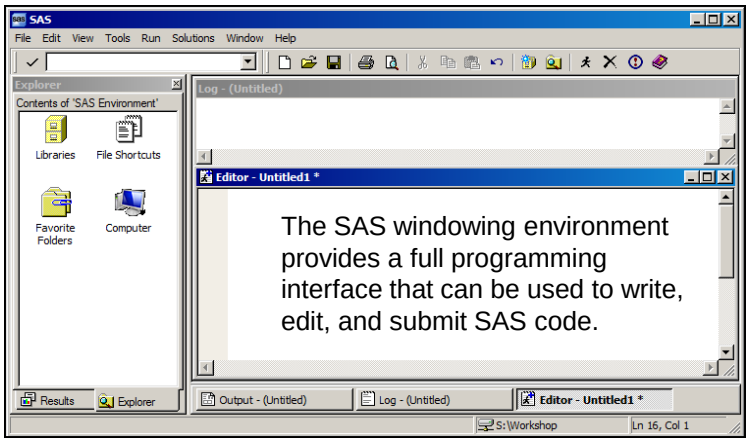
- SAS Windowing Environment
- SAS Enterprise Guide
- SAS Studio

- To enable you to learn more about each of these interfaces, self-directed exercises are available in the course data location.

20

SAS Windowing Environment

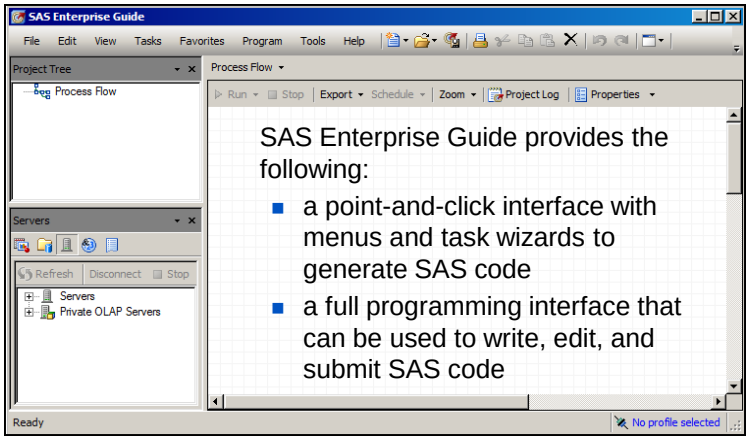
The *SAS windowing environment* is an application that is accessed from different operating environments.



21

SAS Enterprise Guide

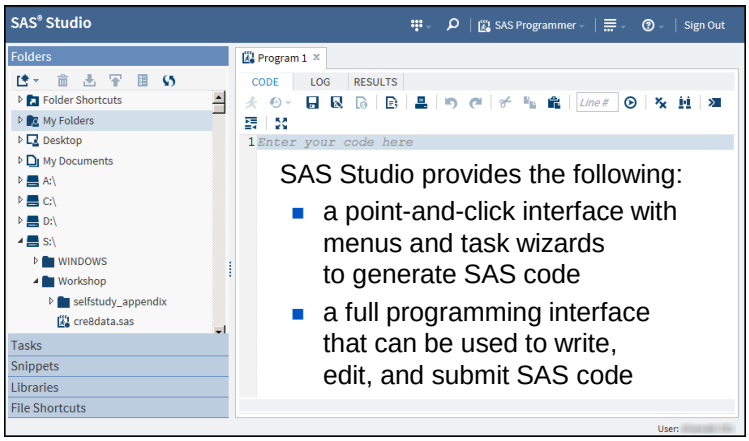
SAS Enterprise Guide is a client application that is accessed from the Windows operating environment.



22

SAS Studio

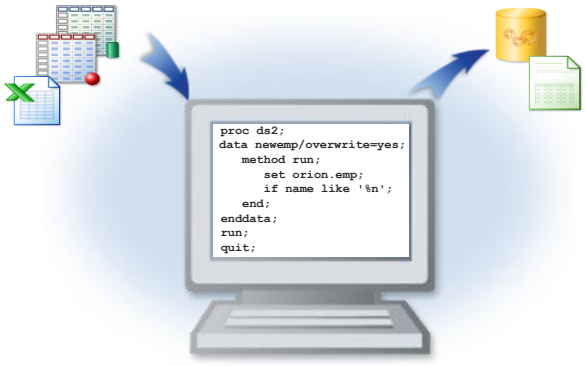
SAS Studio is a web client that is accessed through an HTML5-compliant web browser.



23

Orion Star Business Scenarios

In this course, you ***write*** SAS programs that access and manipulate Orion Star data to create enhanced data sets and reports. You use the editor in one of the three available SAS interfaces.



24

1.02 Multiple Choice Poll

Which SAS interface will you use during this course?

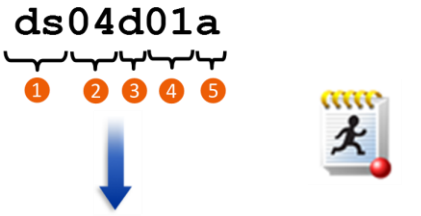
a. SAS Studio
b. SAS Enterprise Guide
c. SAS windowing environment
d. I intend to experiment with multiple interfaces.

25

Program Naming Conventions

In this course, you retrieve and save SAS programs using the structure below.

1. course ID
2. chapter #
3. type
   a=activity
   d=demo
   e=exercise
   s=solution
4. item #
5. placeholder



DS2 Programming, Chapter 4, Demo 1, Part a

26

Three Levels of Exercises

The course is designed so that you complete only ***one*** set of exercises. Select the level that is most appropriate for your skill set.

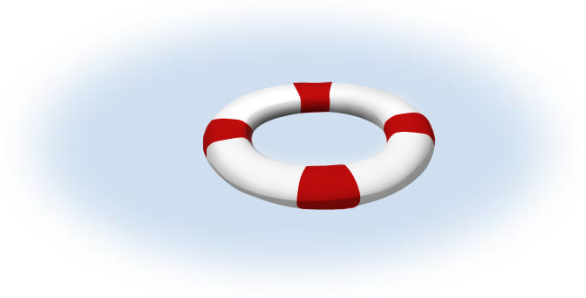
Level 1	Provides step-by-step instructions.
Level 2	Provides less information and guidance.
Challenge	Provides minimal information and guidance. Students might need to use the Help facility.

27

Getting Help

In class, you can get product help in several ways, depending on the editor that is used.

- Help facilities included in the software
- web-based shortcuts on the browser's Favorites bar



28

Extending Your Learning

After class, you have access to an extended learning page that was created for this course. The page includes

- course data and program files
- a PDF file of the course notes
- other course-specific resources.

This page might also be available during class.

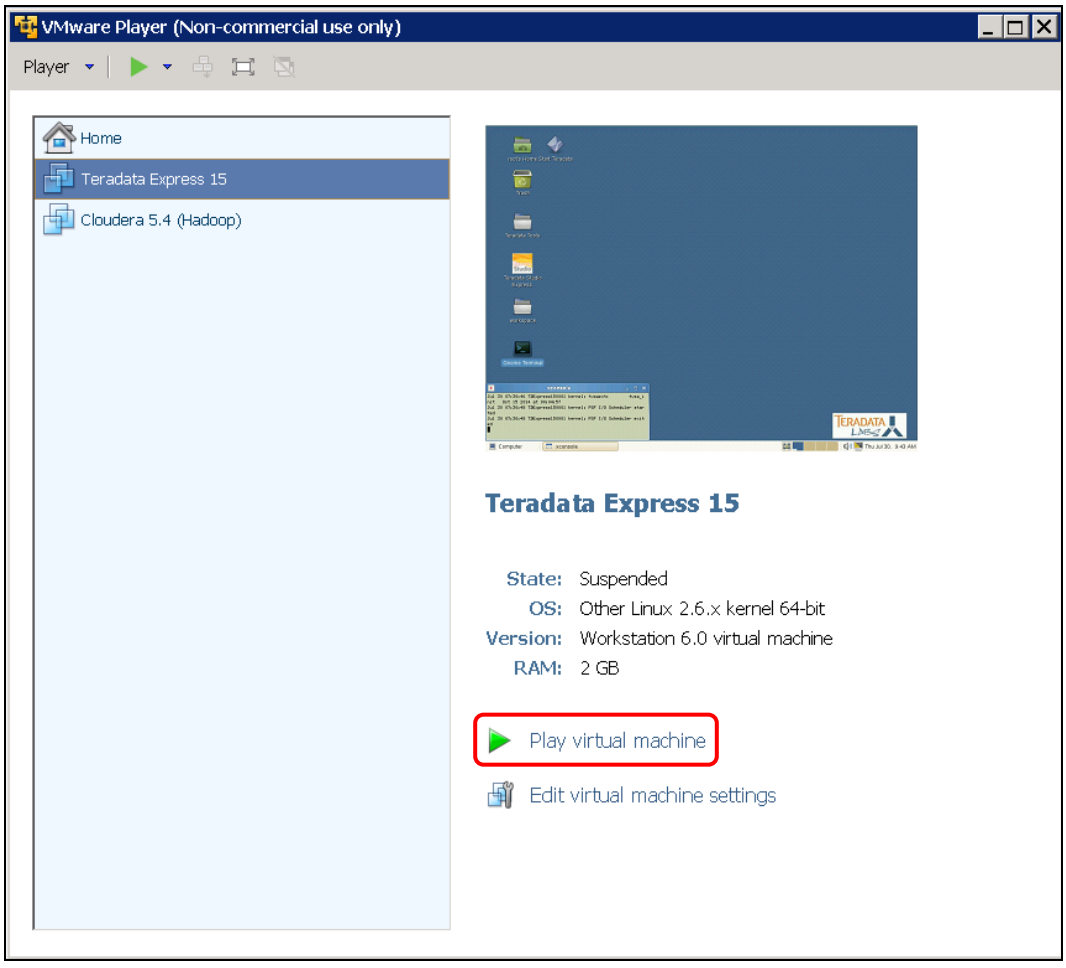
29

Starting and Setting Up the Virtual Lab

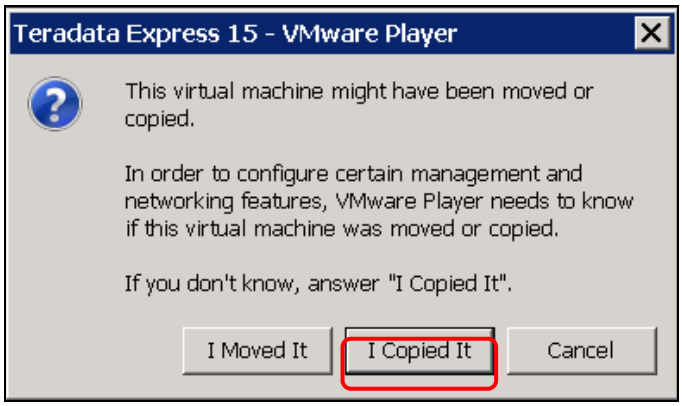
1. Log on to the client desktop. Enter **Student** as the user ID and **Metadata0** as the password. (The password is case sensitive, starts with an uppercase M, and ends with a numeric zero.)

2. From your client machine's Windows desktop, click the **VMware Player** icon on the Windows taskbar.

3. In the VMware Player window, select **Teradata Express 15**. Then click **Play virtual machine.**

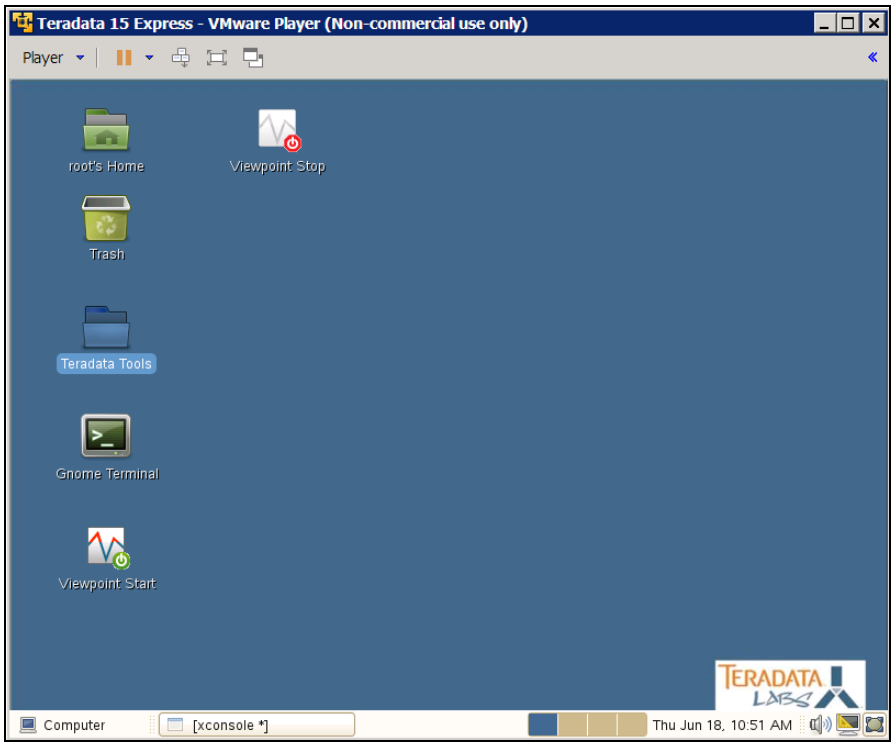


4. You might see a dialog box that asks you whether the virtual machine was moved or copied. Click **I Copied It**.



Clicking **I Moved It** can cause the virtual machine to be unusable.

5. Wait while the machine goes through the resume process. When you see the SUSE Gnome desktop, minimize the VMware Player window. All future work with Teradata is done with SAS.

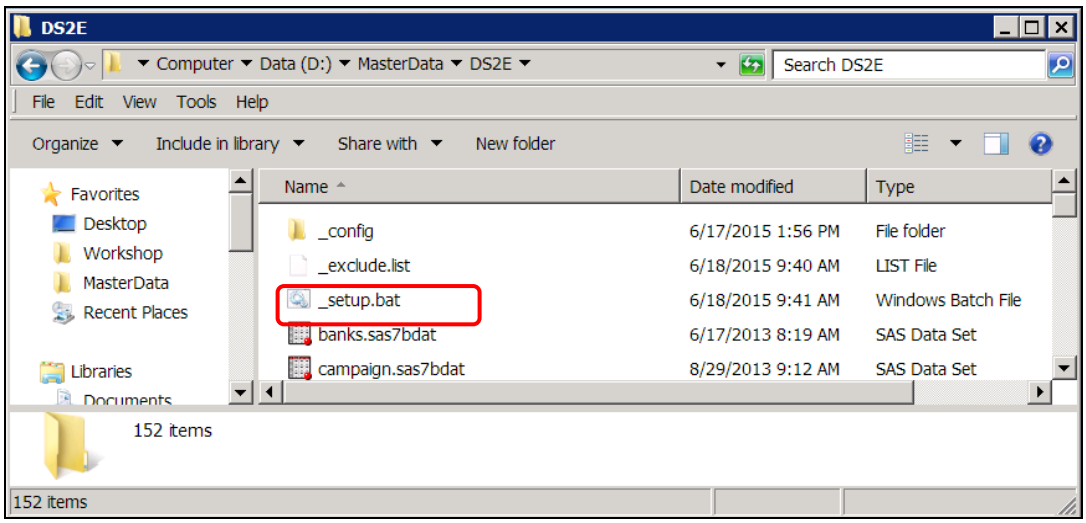


Do ***not*** close the VMware Player window or shut down the virtual machine.

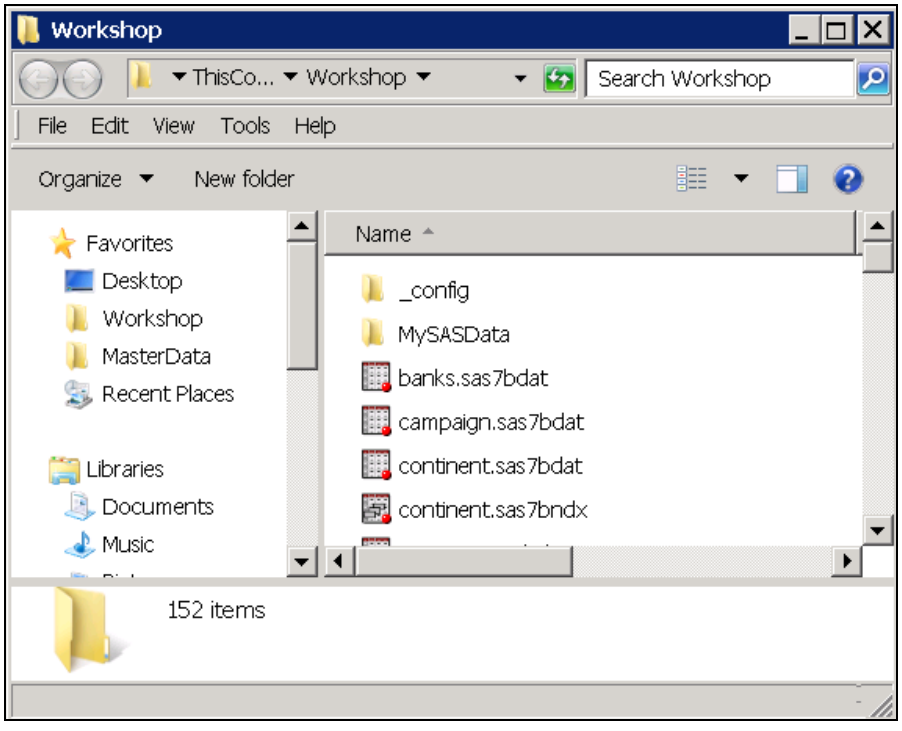
6. Wait approximately two minutes for the Teradata instance to settle before accessing it from SAS. You can perform steps 7 through 9 while you wait.

7. Open Windows Explorer by clicking the icon on the Windows taskbar.

8. Navigate to the **D:\MasterData\DS2E** folder, and double-click the **_setup.bat** file.



9. Wait until the resulting command window closes. Then navigate to the **S:\Workshop** folder and verify that the first few folder names and filenames match the image below to ensure that the setup is complete.

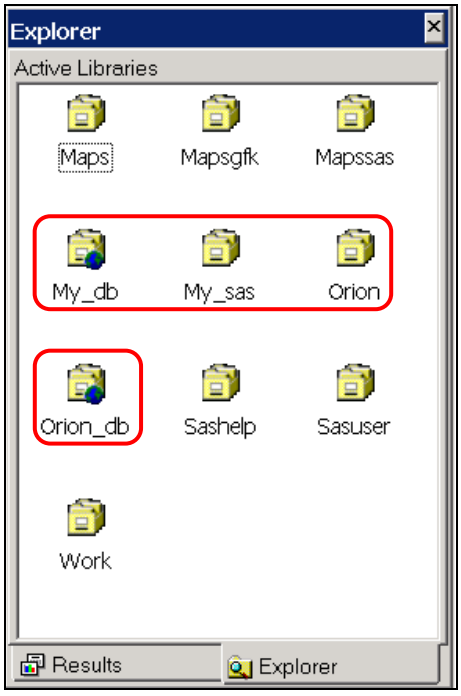


10. Approximately two minutes after you start the Teradata virtual machine, the Teradata instance settles and is ready to use. You should then start your SAS session by clicking either the **SAS 9.4**, **SAS Enterprise Guide**, or **SAS Studio** icon on the Windows taskbar.

11. In the SAS session, open and submit the **s:\Workshop\libnames.sas** program. Verify that there are no errors in the SAS log.

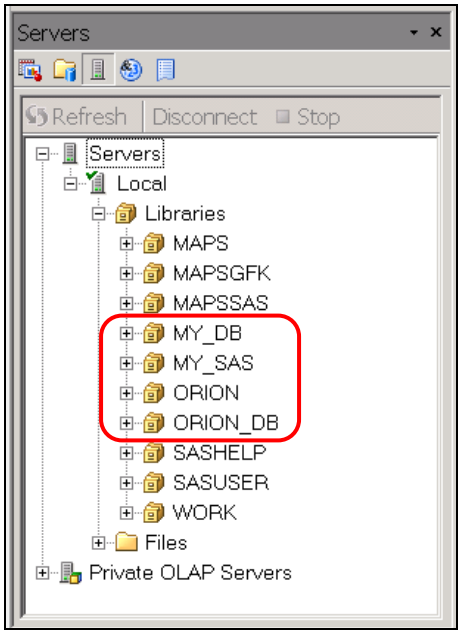
SAS Windowing Environment

Go to the SAS Explorer and ensure that the **My_sas**, **My_db**, **Orion**, and **Orion_db** libraries are listed.



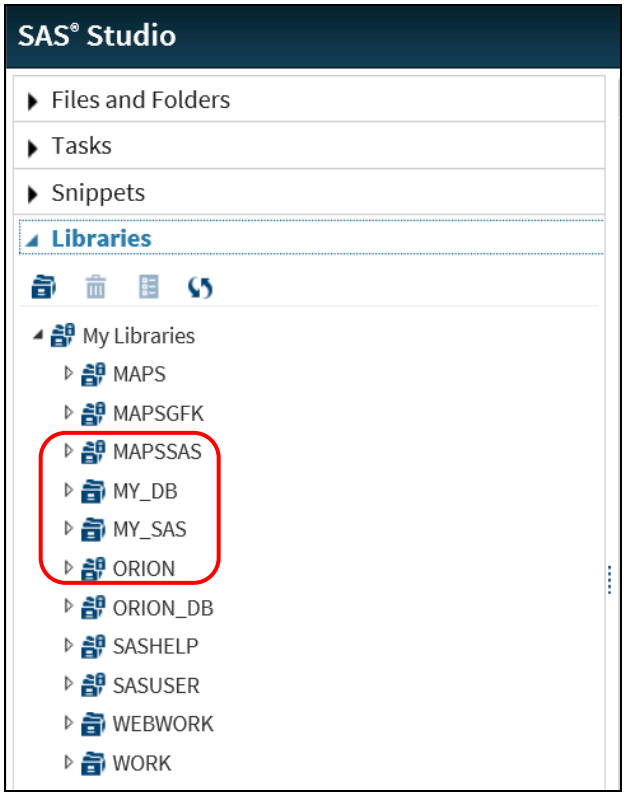
SAS Enterprise Guide

Expand the server list, expand the **Local** server libraries, and verify that the **My_Sas**, **My_Db**, **Orion**, and **Orion_Db** libraries are listed.



SAS Studio

Select the **Libraries** riser bar and verify that the **MY_SAS**, **MY_DB**, **ORION**, and **ORION_DB** libraries are listed.



12. Your Virtual Lab is now ready for you to run activity, demonstration, and exercise programs.

Exercise

1. Preparing the Virtual Lab for Future Activities

Complete all the steps of the previous demonstration in ***your*** Virtual Lab.

1.3 Solutions

Solutions to Exercises

1. Preparing the Virtual Lab for Future Activities

Complete all the steps of the previous demonstration.

Refer to step-by-step instructions for the demonstration.

Chapter 2 Getting Started

2.1 Hello, World

Objectives

- Identify similarities and differences between a DS2 DATA program and a DATA step.
- Submit a DS2 DATA program for execution.

3

Business Scenario

The Orion Star programmers are curious about the syntax similarities and differences between DS2 DATA programs and DATA steps.

4

Hello, World

DS2 Syntax

- related to the DATA step
- but different

```
data _null_;
   Text='Hello, World!';
   put Text=;
run;
```

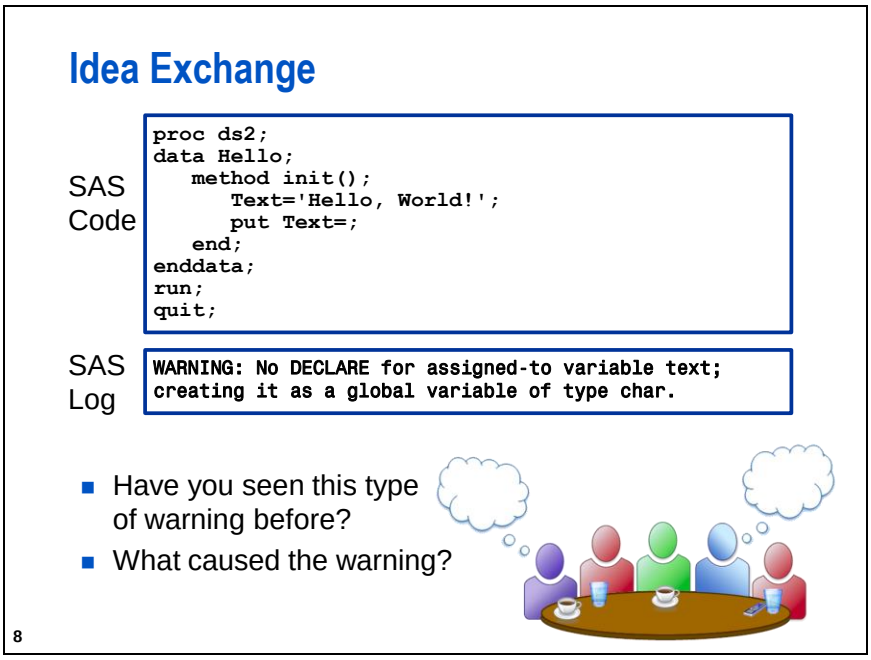
```
proc ds2;
data _null_;
   method init();
      Text='Hello, World!';
      put Text=;
   end;
enddata;
run;
quit;
```

ds02d01

6

A DS2 “Hello, World” Program

1. Open the **ds02d01.sas** program in a SAS Program Editor.
2. Submit Section A - DATA Step and review the log. Notice the values written for the variable **Text** and the expected note that is produced when the program is executed.
3. Submit Section B - DS2 DATA program and review the log. Notice the following:
   - values written for the variable **Text**
   - the unexpected warning about variable declaration
   - the very different note (**Execution succeeded, no rows affected**) that is produced when the program is executed



2.2 Basic DS2 Syntax

Objectives

- Name three DS2 system methods and the conditions under which they execute.
- Describe the functionality of user-defined methods.
- State the purpose and basic syntax of the DS2 DECLARE (DCL) statement.
- Describe the process for creating DS2 variables with global and local scopes.

10

Basic DS2 Syntax

- DS2 includes syntax for three types of programs:
  - DATA programs
  - package programs
  - thread programs
- PROC DS2 uses run-group processing.

```
proc ds2;
package work.pgk;
   <more program statements>
endpackage;
run;
thread work.thread;
   <more program statements>
endthread;
run;
data _null_;
   <more program statements>
enddata;
run;
quit;
```

11

Basic DS2 Syntax

The DS2 DATA program

- begins with a DATA statement
- ends with an ENDDATA statement
- requires a RUN statement to execute.

```
proc ds2;
data _null_ ;
   method init();
      Text='Hello, World!';
      put Text=;
   end;
enddata;
run;
quit;
```

ds02d01

15

Basic DS2 Syntax

New in DS2: Methods

- *Methods* are named, executable blocks of code.
  - The METHOD statement names the method.
  - The END statement terminates the method.
- All executable code is encapsulated in methods.

```
proc ds2;
data _null_;
   method init();
      Text='Hello, World!';
      put Text=;
   end;
enddata;
run;
quit;
```

ds02d01

18

Basic DS2 Syntax

System methods execute automatically.

- INIT() – once at start
- RUN() – once for every row
- TERM() – once at termination

```
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
```

The data set **orion.banks** has three observations. The RUN method executes three times.

Only the RUN method includes an implicit OUTPUT statement.

ds02d02

22

Setup for the Poll

This DS2 DATA program INIT method contains a SET statement that reads **orion.banks**. If the data set contains three observations, how many times is the method executed?

```
proc ds2;
data _null_;
   method init();
      set orion.banks;
      put _all_;
   end;
enddata;
run;
quit;
```

23

2.01 Multiple Choice Poll

This DS2 DATA program INIT method contains a SET statement that reads **orion.banks**. If the data set contains three observations, how many times is the method executed?

a. 0
b. 1
c. 3
d. cannot be determined from the information given

24

Basic DS2 Syntax

```
proc ds2;
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

System methods

- execute automatically
- cannot be explicitly called
- do not accept arguments.

DS2 DATA programs must contain at least one system method.

ds02d02

28

Basic DS2 Syntax

User-defined methods have the following features:

- can accept arguments
- can return a value
- execute when referenced
- can be referenced multiple times

```
proc ds2;
data _null_;
   method c2f(double Tc) returns double;
   /* Celsius to Fahrenheit */
      return (((Tc*9)/5)+32);
   end;
   method init();
      dcl double Degc DegF;
      do DegC=0 to 30 by 15;
         DegF=c2f(DegC);
         PUT DegC= DegF=;
      end;
   end;
enddata;
run;
quit;
```

Method is defined.

Method is called.

Partial SAS Log

```
degc=0  degf=32
degc=15 degf=59
degc=30 degf=86
```

ds02d03

33

Basic DS2 Syntax

Declarative statements come before executable statements.

- Global declaratives affect the entire data program, and are placed before the method definitions.
- Local declaratives affect only the method in which they appear, and are placed before the executable statements.

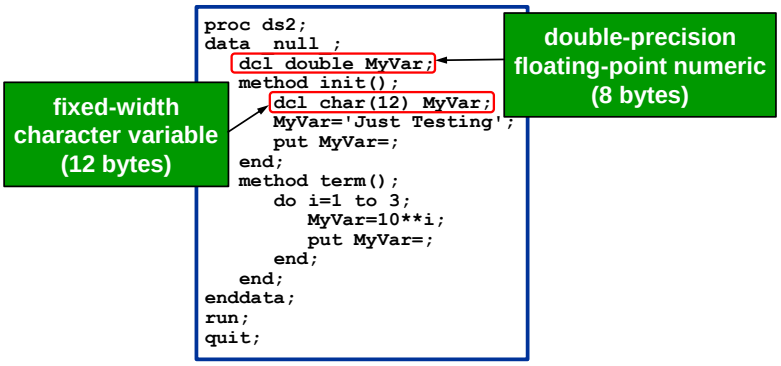
```
proc ds2;
data test;
   dcl double MyVar;
   retain MyVar 0;
   method run();
      dcl integer i;
      set banks;
      do i=1 to 3;
         MyVar=10**i;
      end;
   end;
enddata;
run;
quit;
```

34

Basic DS2 Syntax

The DECLARE statement

- declares variables or temporary arrays
- assigns type, length, and other attributes.



ds02d04

35

Basic DS2 Syntax

General form of the DECLARE statement:

DECLARE *data-type variable-list*
<HAVING LABEL '*string*' | FORMAT | INFORMAT>;

Examples:

```
/* Declare three DOUBLE variables formatted dollar12.2*/
dcl double Var1 Var2 Var3 having format dollar12.2;

/* Declare a high-precision fixed point numeric variable */
dcl decimal(35,5) Var1;

/* Declare a fixed-width character variable labeled 'My Text'*/
dcl char(25) Var1 having label 'My Text';
```

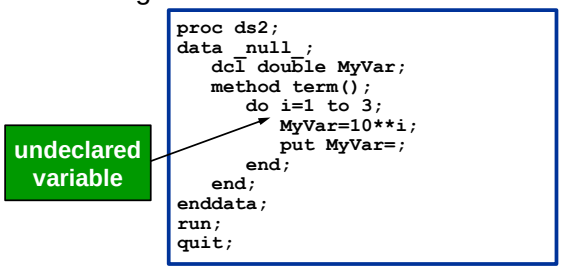
Where a variable is declared in a DS2 program determines the variable's scope.

36

Basic DS2 Syntax

Undeclared Variables

- Variables not in an input data set and not referenced in a DECLARE statement are undeclared.
- The numeric default type is DOUBLE.
- The character default type is CHAR.
- Undeclared variables produce WARNINGs in the SAS log.

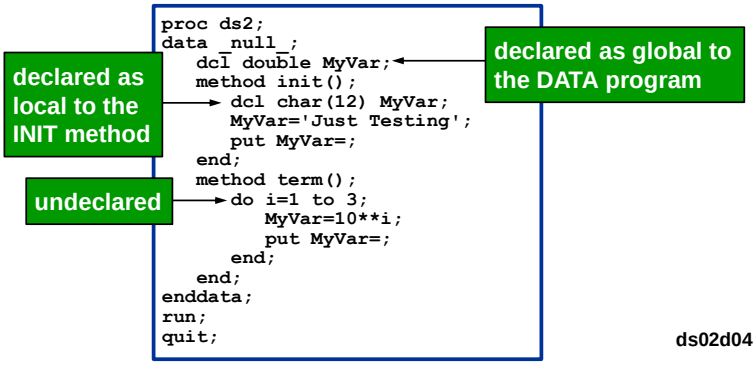


37 ds02d04

Basic DS2 Syntax

Variable Scope

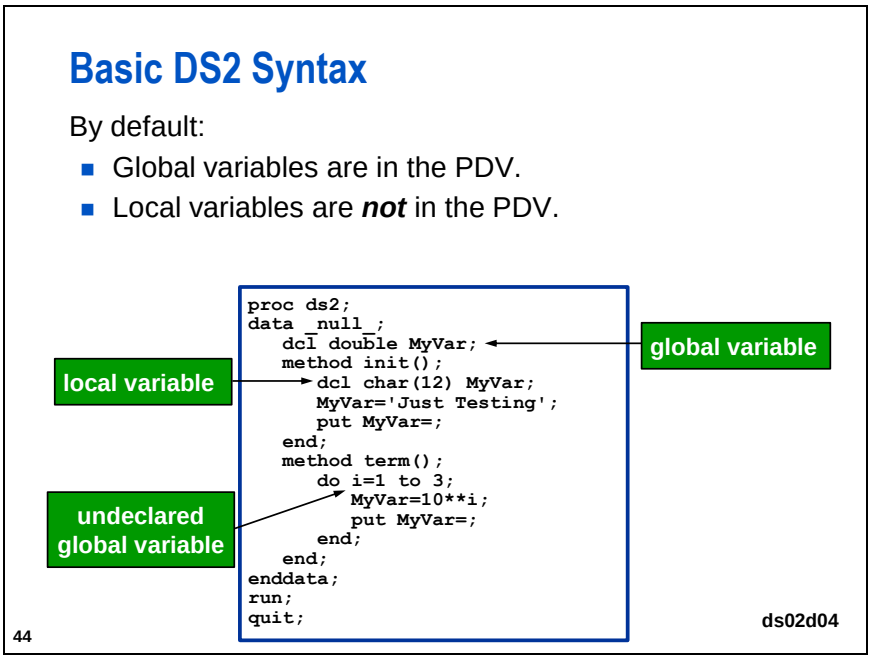
- Locally declared variables have a ***local*** scope.
- Globally declared variables have a ***global*** scope.
- Undeclared variables have a ***global*** scope.



41 ds02d04

The scope within a DS2 DATA program is the following:

- Local identifiers are available only within the method in which they were declared.
- Global identifiers are available throughout the entire DS2 DATA program.



Global variables in the PDV can be excluded from the result set if you use the KEEP or DROP global declarative statements.

Exercises

Level 1

1. Declaring Variables in a DS2 DATA Program

a. Open the **ds02e01**starter program.

```
proc ds2;
data _null_;
   method init();
      put Name=;
   end;
enddata;
run;
quit;
```

b. Modify the **ds02e01** program to assign your name to a variable called **Name** so that your name is written to the SAS log once. Use a DECLARE (DCL) statement to define **Name** as CHAR(30). Run the program and check the SAS log. Verify that your finished program did not produce any warning or error messages.

SAS Log

```
name=Don Johe
NOTE: Execution succeeded. No rows affected.
```

Level 2

2. Reading a Data Set in a DS2 DATA Program

a. Open the **ds02e02** starter program.

```
proc ds2;
data _null_;
   set orion.offices;
   put _n_= Country= City=;
run;
quit;
```

b. Modify the **ds02e02** program. Add the necessary methods to read the **orion.offices** data set and write the values of **_n_**, **Country**, and **City** to the log for all five observations.

Challenge

3. Controlling Variable Scope

a. Open the **ds02e03** starter program.

```
/*ds02e03*/
proc ds2;
data test;

      set orion.offices;
      if Country='US' then Area='North America';
         else Area='Not North America';
      put _n_= Country= Headcount= Area=;

enddata;
run;
quit;

title 'Chapter 2 - Exercise 3';
proc print data=test;
run;
title;
```

b. Modify the program so that all observations of **orion.offices** are processed. Declare any new variables as local so that they are not included in the output data set. Execute your program and verify that no error or warning messages were generated. The SAS log should contain the computed value of **Area** for each observation. The output data set should contain only the **Country**, **City**, and **Headcount** variables.

Hint: If you run the program more than once, you need the OVERWRITE= table option to enable overwriting of the output data set.

Partial SAS Log

```
_N_=1 Country=AU Headcount=26 area=Not North America
_N_=2 Country=AU Headcount=47 area=Not North America
_N_=3 Country=US Headcount=85 area=North America
_N_=4 Country=US Headcount=68 area=North America
_N_=5 Country=US Headcount=82 area=North America
```

PROC PRINT Output

Chapter 2 - Exercise 3

Obs	Country	City	Headcount
1	AU	Melbourne	26
2	AU	Sydney	47
3	US	Miami-Dade	85
4	US	Philadelphia	68
5	US	San Diego	82

2.3 Converting a DATA Step to a DS2 DATA Program

Objectives

- Convert a DATA step to a DS2 DATA program.

47

Converting to DS2

Convert a DATA step to DS2, and leverage the new programming structures and capabilities.

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;









enddata;
run;
quit;
```

ds02d05

48

DS2 uses ANSI quotation marks. Single quotation marks indicate constant text, and double quotation marks indicate an identifier.

Examples:

DS2 DATA Program	DATA Step
`where this_var="That Var";`	`where this_var='That Var'n;`
`where this_var='That Var';`	`where this_var='That Var';` `where this_var="That Var";`

2.02 Quiz

Which DS2 system method should be used to execute sections 1, 2, and 3 of this DATA step?

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

Section	System Method
	INIT
	RUN
	TERM

49

Converting to DS2

Converting Section 1

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;









enddata;
run;
quit;
```

ds02d05

51

Converting to DS2

Converting Section 2

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;

   method run();
      set orion.banks;
      put _all_;
   end;









enddata;
run;
quit;
```

ds02d05

52

Converting to DS2

Converting Section 3

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;

   method run();
      set orion.banks;
      put _all_;
   end;

   method term();
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

ds02d05

53

Converting to DS2

Partial SAS Log

```
**> Starting
NAME=Carolina Bank and Trust     Text=**> Starting _N_=1 RATE=0.0318
NAME=State Savings Bank          Text=             _N_=2 RATE=0.0321
NAME=National Savings and Trust  Text=             _N_=3 RATE=0.0328
**> All done
WARNING: No DECLARE for assigned-to variable text; assuming type char.
NOTE: Execution succeeded. No rows affected.
```

- The variable **Text** is undesirable in the RUN method.
- The **`**> All done`** note is truncated. It is missing the **`!`**.
- Undesired warnings were generated in the SAS log.

ds02d05

54

Setup for the Poll

What is the scope of the undeclared variable **Text**?

```
proc ds2;
data _null_;
   method init();
      Text='**> Starting';
      put Text;
   end;

   method run();
      set orion.banks;
      put _all_;
   end;

   method term();
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```

ds02d05

55

2.03 Multiple Choice Poll

What is the scope of the undeclared variable **Text**?

a. local
b. global
c. cannot be determined

56

Converting to DS2

Declaring the variable **Text** as local excludes it from the PDV.

```
proc ds2;
data _null_;
   method init();
      dcl char(12) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(13) Text;
      Text='**> All done!';
      put Text;
   end;
run;
enddata;
quit;
```

ds02d05

58

Converting to DS2

Partial SAS Log

```
**> Starting
NAME=Carolina Bank and Trust     _N_=1 RATE=0.0318
NAME=State Savings Bank          _N_=2 RATE=0.0321
NAME=National Savings and Trust  _N_=3 RATE=0.0328
**> All done!
NOTE: Execution succeeded. No rows affected.
```

- The variable **Text** does not exist in the RUN method.
- The `**> All done!` note is no longer truncated.
- No errors or warnings were generated.

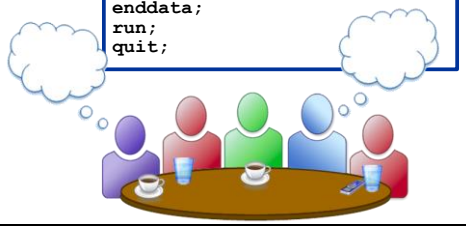
ds02d05

59

Idea Exchange

What real-world benefit do you see resulting from converting this particular program from a DATA step to a DS2 DATA program?

```
proc ds2;
data _null_;
   method init();
      dcl varchar(20) Text;
      Text='**> Starting';
      put Text;
   end;
   method run();
      set orion.banks;
      put _all_;
   end;
   method term();
      dcl char(11) Text;
      Text='**> All done!';
      put Text;
   end;
enddata;
run;
quit;
```



60

Review: When to Choose DS2

DS2 programs are best suited for applications that do the following:

- take advantage of threaded processing
  - are computationally complex
  - can execute in massively parallel processing (MPP) databases
- require the precision that the DS2 data types offer
- leverage the convenient reusability of the DS2 modules and packages

61

Exercises

Level 1

4. Converting a DATA Step Program to a DS2 DATA Program

a. Open the **ds02e04** program and review the DATA step code.

b. Convert this program to a DS2 DATA program that produces the same values in the SAS log without producing error or warning messages.

SAS Log (original DATA step)

```
Country=AU City=Melbourne Headcount=26 Total=26
Country=AU City=Sydney Headcount=47 Total=73
Country=US City=Miami-Dade Headcount=85 Total=158
Country=US City=Philadelphia Headcount=68 Total=226
Country=US City=San Diego Headcount=82 Total=308
NOTE: There were 5 observations read from the data set ORION.OFFICES.
```

Desired SAS Log (DS2 DATA program)

```
Country=AU City=Melbourne    Headcount=26 Total=26
Country=AU City=Sydney       Headcount=47 Total=73
Country=US City=Miami-Dade   Headcount=85 Total=158
Country=US City=Philadelphia Headcount=68 Total=226
Country=US City=San Diego    Headcount=82 Total=308
NOTE: Execution succeeded. No rows affected.
188  QUIT;
```

Level 2

5. Leveraging User-Defined Methods

a. The **orion.banks** data set contains three rows. Each row contains the name of a bank and the annual interest rate that each bank offers on an investment account. Interest compounds quarterly for each of the investment accounts. You must determine how much interest would be paid by each bank on a one-year investment of $50,000. Open the **ds02e05** starter program and modify the DS2 program.

1) Write a user-defined method named IntPaid() that accepts two arguments (Amount, Rate) and returns the total interest to be paid for a year, based on quarterly compounding of interest.

2) Add the required system methods so that

   a) all rows of **orion.banks** are read

   b) the IntPaid() user-defined method is called to calculate the interest earned, based on the interest rate paid by each bank

   c) the name of the bank and the total interest earned are written to the SAS log.

b. Calculate the interest based on one of these two processes:

- iterative processing algorithm

```
do Q=1 to 4;
   Interest+SUM(Interest,Amount)*(Rate/4);
```

```
end;
```

- compound interest algorithm

```
Interest=round(Amount*(1+Rate/4)**4 - Amount,.01);
```

SAS Log

```
name=Carolina Bank and Trust    annualinterest=1609.06
name=State Savings Bank         annualinterest=1624.42
name=National Savings and Trust annualinterest=1660.28
NOTE: Execution succeeded. No rows affected.
```

2.4 Solutions

Solutions to Exercises

1. **Declaring Variables in a DS2 DATA Program**

```
proc ds2;
data _null_;
   method init();
      dcl char(30) Name;
      name='Don Johe';
      put Name=;
   end;
run;
quit;
```

2. **Reading a Data Set in a DS2 DATA Program**

```
proc ds2;
data _null_;
   method run();
      set orion.offices;
      put _n_= Country= City=;
   end;
endata;
run;
quit;
```

3. **Controlling Variable Scope**

```
proc ds2;
data test (overwrite=yes);
   method run();
      dcl varchar(20) area;
      set orion.offices;
      if Country='US' then Area='North America';
         else Area='Not North America';
      put _n_= Country= Headcount= Area=;
   end;
enddata;
run;
quit;
title 'Chapter 2 - Exercise 3';
proc print data=test;
run;
title;
```

4. Converting a DATA Step Program to a DS2 DATA Program

```
proc ds2;
data _null_;
   /* Total must be declared GLOBAL in order to RETAIN values */
   dcl double Total;
   method run();
      set orion.offices;
      Total+Headcount;
      Put Country= City= Headcount= Total=;
   end;
enddata;
run;
quit;
```

5. Leveraging User-Defined Methods

Solution Alternative 1: Using a DO Loop

```
proc ds2;
data _null_;
   method IntPaid(double Amount, double Rate) returns double;
      dcl integer Q;
      dcl double Interest;
      do Q=1 to 4;
         Interest+SUM(Interest,Amount)*(Rate/4);
      end;
      return round(Interest,.01);
   end;
   method run();
      dcl double AnnualInterest;
      set orion.banks;
      AnnualInterest=IntPaid(50000,Rate);
      put Name= AnnualInterest=;
   end;
enddata;
run;
quit;
```

Solution Alternative 2: Using a Compound Interest Formula

```
proc ds2;
data _null_;
   method IntPaid(double Amount, double Rate) returns double;
      return round(50000*(1+Rate/4)**4 - 50000,.01);
   end;
   method run();
      dcl double AnnualInterest;
      set orion.banks;
      AnnualInterest=IntPaid(50000,Rate);
      put Name= AnnualInterest=;
   end;
enddata;
run;
quit;
```

Solutions to Student Activities (Polls/Quizzes)

2.01 Multiple Choice Poll – Correct Answer

This DS2 DATA program INIT method contains a SET statement that reads **orion.banks**. If the data set contains three observations, how many times is the method executed?

```
proc ds2;
data _null_;
   method init();
      set orion.banks;
      put _all_;
   end;
enddata;
run;
quit;
```

a. 0
b. 1
c. 3
d. cannot be determined from the information given

The INIT system method automatically executes only once, when the DS2 DATA program first begins execution.

25

2.02 Quiz – Correct Answer

Which DS2 system method should be used to execute sections 1, 2, and 3 of this DATA step?

```
data _null_;
   /* Section 1 */
   if _n_ =1 then do;
      Text='**> Starting';
      put Text;
   end;

   /* Section 2 */
   set orion.banks end=last;
   put _all_;

   /* Section 3 */
   if last then do;
      Text='**> All done!';
      put Text;
   end;
run;
```

Section	System Method
1	INIT
2	RUN
3	TERM

50

2.03 Multiple Choice Poll – Correct Answer

What is the scope of the undeclared variable **Text**?

a. local
b. global
c. cannot be determined

Undeclared variables have a global scope.

57

Chapter 3 DATA Steps versus DS2 DATA Programs

3.1 Similarities to the DATA Step

Objectives

- Identify DS2 language statements that operate in the same manner as their DATA step counterparts.
- Construct a functional DS2 DATA program that contains executable statements.
- List six subtle differences between DATA step and DS2 DATA program processing defaults.

3

Similar to a DATA Step

Many statements have the same functionality in a DS2 DATA program as they do in a DATA step:

- SET
- MERGE
- BY-group processing with First. and Last.
- DO groups and loops
  - DO I= start TO stop BY interval
  - DO WHILE () or DO UNTIL ()
  - loop flow control with CONTINUE, LEAVE, END
- program flow control with RETURN, GOTO, STOP, OUTPUT
- declarative statements KEEP, DROP, RETAIN, RUN (but valid only outside METHOD definitions)

4 ...

In a DS2 DATA program, compile-time statements (such as KEEP, DROP, DECLARE, and so on) must appear in the program code before the first METHOD statement.

Similar to a DATA Step, But...

1.

```
proc ds2;
data newbanks;


      set orion.banks;


enddata;
run;
quit;
```

```
ERROR: Compilation error.
ERROR: Parse encountered SET when expecting end of input.
ERROR: Parse failed on line 3:  >>> set <<<  orion.banks;
```

ds03d01

5

Similar to a DATA Step, But...

1. Executable code must be part of a METHOD.

```
proc ds2;
data newbanks;

   method run();
      set orion.banks;

   end;
enddata;
run;
quit;
```

```
NOTE: Execution succeeded. 3 rows affected
```

ds03d01

6

Similar to a DATA Step, But...

2.

```
proc ds2;
data newbanks;

   method run();
      set orion.banks;

   end;
enddata;
run;
quit;
```

```
ERROR: Compilation error.
ERROR: Base table or view already exists NEWBANKS
ERROR: Unable to execute CREATE TABLE statement for table
WORK.NEWBANKS.
```

ds03d01

7

Similar to a DATA Step, But...

2. Data is ***not*** overwritten by default.

```
proc ds2;
data newbanks /overwrite=yes;

   method run();
      set orion.banks;

   end;
enddata;
run;
quit;
```

```
NOTE: Execution succeeded. 3 rows affected
```

ds03d01

8

Similar to a DATA Step, But...

3.

```
proc ds2;
data newbanks/overwrite=yes;

   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

```
WARNING: Line 976: No DECLARE for assigned-to variable total;
creating it as a global variable of type double.
```

ds03d01

9

Similar to a DATA Step, But...

3. New variables are expected to be declared.

```
proc ds2;
data newbanks/overwrite=yes;
   dcl double Total;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

```
NOTE: Execution succeeded. 3 rows affected
```

ds03d01

10

Strict Variable Declaration

The DS2SCOND= system option and the PROC DS2 SCOND option control handling of undeclared variables.

WARNING	Default: Write a warning to the SAS log.
NONE	No action: The SAS log is not affected.
NOTE	Write a note to the SAS log.
ERROR	Write a compilation error to the SAS log; stop compiling this step.

```
options ds2scond=error;
proc ds2 scond=none;
...
quit;
```

11

The PROC DS2 SCOND= option overrides the DS2SCOND system option.

Strict Variable Declaration

1. Open the **ds03d02** program. Submit the OPTIONS statement and DS2 step. The latter contains an undeclared variable (**Total**). Notice the ERROR message that is produced in the SAS log:

```
options DS2SCOND=ERROR;
proc ds2;
data newbanks/overwrite=yes;
   method  run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

SAS Log

```
1    options DS2SCOND=ERROR;
2    proc ds2;
3    data newbanks/overwrite=YES;
4       method  run();
5          set orion.banks;
6          Total=sum(Rate,.05)*50000;
7       end;
8    enddata;
9    run;
ERROR: Compilation error.
ERROR: Line 6: No DECLARE for assigned-to variable total; creating it as a global variable
       of type double.
10   quit;
```

2. With the DS2SCOND=ERROR system option still in effect, use the PROC DS2 SCOND= option to override this step. The Step 2 code should execute without producing any messages about the undeclared variable:

```
proc ds2 scond=NONE;
data newbanks/overwrite=yes;
   method  run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

SAS Log

```
11   proc ds2 scond=NONE;
12   data newbanks/overwrite=YES;
13      method  run();
14         set orion.banks;
15         Total=sum(Rate,.05)*50000;
16      end;
17   enddata;
18   run;
```

```
NOTE: Execution succeeded. 3 rows affected.
19   quit
```

3. Submit the OPTIONS DS2SCOND=WARNING statement and DS2 DATA program in Step 3. DS2SCOND=WARNING is the default setting for this option. Notice the customary DS2 undeclared variable warning message in the SAS log.

```
options DS2SCOND=WARNING;
proc ds2;
data newbanks/overwrite=yes;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

SAS Log

```
20   options DS2SCOND=WARNING;
21   proc ds2;
22   data newbanks/overwrite=YES;
23      method run();
24         set orion.banks;
25         Total=sum(Rate,.05)*50000;
26      end;
27   enddata;
28   run;
WARNING: Line 25: No DECLARE for assigned-to variable total; creating it as a global variable
         of type double.
NOTE: Execution succeeded. 3 rows affected.
29   quit;
```

4. With the DS2SCOND=WARNING system option still in effect, use the PROC DS2 SCOND option to override Step 4. Execution should produce a note instead of the undeclared variable warning.

```
proc ds2 SCOND=NOTE;
data newbanks/overwrite=yes;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

SAS Log

```
30   proc ds2 SCOND=NOTE;
31   data newbanks/overwrite=yes;
32      method run();
33         set orion.banks;
34         Total=sum(Rate,.05)*50000;
35      end;
36   enddata;
37   run;
NOTE: Line 34: No DECLARE for assigned-to variable total; creating it as a global variable
```

```
      of type double.
NOTE: Execution succeeded. 3 rows affected.
38   quit;
```

5. With the DS2SCOND=WARNING system option still in effect, modify the DS2 DATA program to declare the variable **Total**. Step 5 should execute without producing any undeclared variable messages.

```
proc ds2;
data newbanks/overwrite=yes;
   dcl double Total;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

SAS Log

```
39   proc ds2;
40   data newbanks/overwrite=yes;
41      dcl double Total;
42      method run();
43         set orion.banks;
44         Total=sum(Rate,.05)*50000;
45      end;
46   enddata;
47   run;
NOTE: Execution succeeded. 3 rows affected.
48   quit;
```

Similar to a DATA Step, But...

4.

```
proc ds2;
data newbanks /overwrite=yes;

   method run();
      set orion.banks;
      put Name @30 Rate percent5.1;
   end;
enddata;
run;
quit;
```

```
ERROR: Compilation error.
ERROR: Parse encountered invalid character when expecting ';'.
ERROR: Line xxx: Parse failed: put Name  >>> @ <<< 30 Rate
percent5.1;
```

ds03d01

13

Similar to a DATA Step, But...

4. PUT statement column and line parameters are not supported.

```
proc ds2;
data newbanks /overwrite=yes;
   dcl double total;
   method run();
      set orion.banks;
      put Name $28. Rate percent5.1;
   end;
enddata;
run;
quit;
```

```
NOTE: Execution succeeded. 3 rows affected
```

ds03d01

14

Similar to a DATA Step, But...

5.

```
proc ds2;
data data /overwrite=yes;
   dcl double total;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

```
ERROR: Compilation error.
ERROR: Parse encountered DATA when expecting ';'.
ERROR: Parse failed on line 2: data  >>> data <<<  /overwrite=yes;
```

ds03d03

15

Similar to a DATA Step, But...

5. Keywords are ***reserved*** words.

```
proc ds2;
data "data" /overwrite=yes;
   dcl double total;
   method run();
      set orion.banks;
      Total=sum(Rate,.05)*50000;
   end;
enddata;
run;
quit;
```

```
NOTE: Execution succeeded. 3 rows affected
```

DS2 reserved words are listed in the online documentation.

ds03d03

16

Reserved words used as identifiers must be quoted using the ANSI standard double quotation marks. A complete list of reserved words can be found in the *SAS® 9.4 DS2 Language Reference* in the DS2 Concepts – DS2 Reserved Words category.

Similar to a DATA Step, But...

6. DS2 uses ANSI quoting standards.

```
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"='This is my text';
      put "My Var";
   end;
enddata;
run;
quit;
```

```
This is my text
NOTE: Execution succeeded. 0 rows affected
```

ds03d04

17

ANSI Quotation Marks

- Double quotation marks delimit identifiers (object names).
- Single quotation marks delimit character literals.

18 ...

ANSI Quotation Mark Behavior in DS2

Identifiers containing special characters are enclosed in double quotation marks.

```
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"='This is my text';
      put "My Var";
   end;
enddata;
run;
quit;
```

```
This is my text
NOTE: Execution succeeded. 0 rows affected
```

ds03d04

19

ANSI Quotation Mark Behavior in DS2

Character constants are enclosed in single quotation marks.

```
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"='This is my text';
      put "My Var";
   end;
enddata;
run;
quit;
```

```
This is my text
NOTE: Execution succeeded. 0 rows affected
```

ds03d04

20

ANSI Quotation Mark Behavior and SAS Macro

Single quoted literals do not resolve macro variables.

```
%let macvar=This is my text;
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"='&macvar';
      put "My Var";
   end;
enddata;
run;
quit;
```

```
&macvar
NOTE: Execution succeeded. 0 rows affected
```

ds03d04

21

ANSI Quotation Mark Behavior and SAS Macro

Double quoted strings are interpreted as identifiers.

```
%let macvar=This is my text;
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"="&macvar";
      put "My Var";
   end;
enddata;
run;
quit;
```

The undeclared numeric variable **This is my text** contains a missing value.

```
.
WARNING: Line 30: No DECLARE for referenced variable "this is
my text"; assuming type double.
```

22 ds03d04

ANSI Quotation Mark Behavior and SAS Macro

The **%tslit** macro places single quotation marks around the text resolved from a macro variable reference.

```
%let macvar=This is my text;
proc ds2;
data _null_;
   method init();
      dcl varchar(15) "My Var";
      "My Var"=%tslit(&macvar);
      put "My Var";
   end;
enddata;
run;
quit;
```

%tslit(*text value*)

```
This is my text
NOTE: Execution succeeded. 0 rows affected
```

23 ds03d04

The “Subtle Six” Dissimilarities

1. Executable code must be part of a METHOD definition.
2. Data sets are not overwritten by default.
3. New variables should be declared.
4. PUT statement column and line parameters are not supported.
5. Keywords are ***reserved*** words.
6. ANSI quoting standards are as follows:
   - Double quotation marks delimit identifiers.
   - Single quotation marks delimit text literals.
   - Use **%tslit** to single-quote resolved macro text.

24

Exercises

Level 1

1. Reviewing an Undeclared Variable

a. Open the **ds03e01** program. Submit the program and review the SAS log. Notice the warning concerning the undeclared variable **Address**.

b. Add the SCOND=NONE option to the PROC DS2 statement. Submit the program and verify that no warnings or error messages are produced.

c. Modify the DATA _NULL_ statement to create the output data set **work.mailing**. Uncomment the PROC CONTENTS step at the end of the program. Submit the program and verify that no warnings or error messages are produced. Review the PROC CONTENTS output. What is the length of the variable **Address**? _____ bytes

d. Add a global DECLARE statement to the program. Declare the variable **address** as char(50). Remove the SCOND=NONE option from the PROC DS2 statement and add the /OVERWRITE=YES option to the DATA statement. Submit the program and review the SAS log. Verify that no warnings or error messages are produced. Then review the PROC CONTENTS output. What is the length of the variable **Address**? _____ bytes

Level 2

2. Modifying a Program to Include an Undeclared Variable

a. Open the **ds03e02** program. Submit the program and review the SAS log. Notice the warning that the variable **Threshold** is not declared.

b. Review the report. Notice that many rows with salaries less than the threshold value are included in the **higher** data set and that all values for the **Threshold** variable are missing.

c. Modify the program so that the **Threshold** variable value is retained and included in the output data set.

Hint: Only globally declared variables are included in the PDV and output data set.

Challenge

3. Converting a DATA Step and Examining the Results

a. Open the **ds03e03** program. Submit the program.

b. How many observations are in the **transaction** data set that is used to produce the report titled "Cash or Check transactions"? _____

c. Uncomment the second %LET statement and resubmit the program. How many observations are in the **transaction** data set that is used to produce the report titled "Credit Card transactions"?

_____

d. Convert the DATA step to a DS2 DATA program. Resubmit the program twice. Produce both the "Credit Card transactions" and "Cash or Check transactions" **transaction** data sets and associated reports. Verify that the modified program produces no errors or warnings in the SAS log.

3.2 DS2 “Missing” Features

Objectives

- Write DS2 DATA program code to do the following:
  - subset variables and rows
  - conduct iterative processing using arrays and DO loops
  - control variable attributes
- Identify DATA step statements that have no DS2 counterparts.

27

DATA Steps versus DS2 DATA programs

There are differences in the languages.

- DATA step syntax includes many functions and statements that are not found in DS2.
- DS2 has several functions and statements that have no DATA step counterparts.

Is any DATA step functionality missing in DS2?

28

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement
WHERE

```
data test1;
   set orion.continent;
   where Continent_ID > 93;
run;
```

29 ds03d10

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
WHERE	FedSQL query with WHERE

```
data test1;
   set orion.continent;
   where Continent_ID > 93;
run;
```

```
proc ds2;
data test2 /overwrite=yes;
   method run();
      set
     {select *
         from orion.continent
         where Continent_ID>93};
   end;
enddata;
run;
quit;
```

30 ds03d10

FedSQL

- DS2 leverages FedSQL, a new, modernized SAS proprietary SQL implementation that features the following:
  - ANSI SQL:1999 standards
  - improved ANSI compliance
  - support for ANSI data types
  - scalable, threaded, high-performance
  - improved implicit pass-through
- FedSQL query results can be used as input to a DS2 SET statement.
- Use PROC FedSQL to execute FedSQL queries without invoking DS2.

31

FedSQL

- FedSQL includes a few SAS extensions.
  - INFORMAT/FORMAT/LABEL column modifiers
    - only in CREATE TABLE statements for SAS, SPD Engine, or SASHDAT tables
    - only for CHAR or DOUBLE columns
  - table options primarily to improve performance with DBMS data
  - uses Base SAS, FCMP functions, and DS2 methods
  - no SAS environment dictionary tables
  - richer data source dictionary tables

32 ...

FedSQL

FedSQL Dictionary Table Details

Dictionary Table	Description
CATALOGS	Catalog driver and description information
COLUMNS	Detailed column information for all tables, including actual ANSI data types
COLUMN_STATISTICS	Table and index statistic-based columns, such as cardinality, common values, and null fraction
STATISTICS	Table and index statistics such as creating and modifying dates, and partitioning and fragmentation.
TABLES	Table name, associated catalog or schema, and table type (table, view, alias, and so on)

33 ...

See the PROC FEDSQL documentation for more information.

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement
UPDATE, MODIFY

```
proc sql;
create table updates as
select Product_ID
     , New_Price as Price
   from my_db.price_updates
   order by Product_id
;
quit;
data my_sas.prices;
   update my_sas.prices
          updates;
   by Product_id;
run;
```

ds03d05

34

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
UPDATE, MODIFY	The SQLSTMT package

```
proc sql;
create table updates as
select Product_ID
     , New_Price as Price
   from my_db.price_updates
   order by Product_id
;
quit;
data my_sas.prices;
   update my_sas.prices
          updates;
   by Product_id;
run;
```

```
data _null_;
method run();
   dcl package sqlstmt updt
       ('update my_sas.prices
           set Price=?
           where Product_ID=?'
     ,[New_Price Product_ID]);
   set {select product_ID
              , New_Price
     from my_db.price_updates};
   updt.execute();
end;
enddata;
run;
quit;
```

ds03d05

35

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
ATTRIB, LABEL, LENGTH, FORMAT, INFORMAT	DECLARE

```
data test1;
   length A $4;
   attrib A informat=$CHAR4.;
   format B comma10.2;
   label A='Text' B='Number';
   a=' xx ';
   b=10000.00000;
run;
```

Test1		Test2	
Text	Number	Text	Number
xx	10,000.00	xx	10,000.00

```
proc ds2;
data test2 /overwrite=yes;
   dcl char(4) A having
       informat $char4.
       label 'Text';
   dcl double B having
       format comma10.2
       label 'Number';
   method run();
      a=' xx ';
      b=10000.00000;
   end;
enddata;
run;
quit;
```

Only one statement syntax to remember!

ds03d06

37

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
ARRAY (referencing PDV variables)	VARARRAY

```
data test1;
   array N [5];
   array C [5] $ 1 _temporary_;
   do i=1 to dim(n);
      N[i]=dim(n)-I;
   end;
   do i=1 to dim(c);
      C[i]=BYTE(64+I);
      put C[i]= @;
   end;
   put _all_;
run;
```

```
proc ds2;
data test2/overwrite=yes;
   vararray double N[5];
   dcl char(1) C[5];
   method run();
      dcl integer i;
      do i=1 to dim(n);
         N[i]=dim(n)-I;
      end;
      do i=1 to dim(c);
         C[i]=BYTE(64+I);
      end;
      put C[*]=;
      put _all_;
   end;
enddata;
run;
quit;
```

VARARRAY is a ***global*** statement. It is not valid inside a method.

39 ds03d07

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
ARRAY (_temporary_ array)	DECLARE

```
data test1;
   array N [5];
   array C [5] $ 1 _temporary_;
   do i=1 to dim(n);
      N[i]=dim(n)-I;
   end;
   do i=1 to dim(c);
      C[i]=BYTE(64+I);
      put C[i]= @;
   end;
   put _all_;
run;
```

```
proc ds2;
data test2/overwrite=yes;
   vararray double N[5];
   dcl char(1) C[5];
   method run();
      dcl integer i;
      do i=1 to dim(n);
         N[i]=dim(n)-I;
      end;
      do i=1 to dim(c);
         C[i]=BYTE(64+I);
      end;
      put C[*]=;
      put _all_;
   end;
enddata;
run;
quit;
```

Temporary arrays are declared with a DECLARE statement, and the elements are ***never*** PDV variables.

40 ds03d07

DS2 Arrays

Use the DECLARE statement to create a temporary array.

```
proc ds2;
data _null_;
   dcl smallint T1[3:5];
   method init();
      dcl double T2[3,3];
      dcl integer I J;
      T1:=(3,2,1);
      do i=3 to 5;
         put t1[I]=;
         do j=1 to 3;
            t2[i-2,j]=i-j;
         end;
      end;
      put 'T2: ' T2[*];
      put 'PDV Variables';
      put _all_;
   end;
enddata;
run;
quit;
```

DECLARE *data-type array-name*
[*dim-lower*:*dim-upper* | *a*<,*b* … ,*n*>]

```
t1[3]=3
t1[4]=2
t1[5]=1
T2: 2 1 0 3 2 1 4 3 2
PDV Variables
_n_=1
```

ds03d08a

42

DS2 ARRAY Assignment Statement

The VARARRAY statement creates an array of PDV variables.

```
proc ds2;
data _null_;
   vararray double Q[4] Qtr1-Qtr4;
   method init();
      set {select Employee_ID
                  , Qtr1, Qtr2, Qtr3, Qtr4
             from orion.Employee_Donations};
      put 'Q:  ' Q[*];
      put 'PDV Variables';
      put _all_;
   end;
enddata;
run;
quit;
```

VARARRAY *data-type array-name*
[*dim-lower*:*dim-upper* | *a*<,*b* … ,*n*>|*]
<varlist>

```
Q:  . . . 25
PDV Variables
EMPLOYEE_ID=120265 Qtr1=. Qtr2=. _N_=1 Qtr3=. Qtr4=25
```

ds03d08b

41

VARARRAY is not valid in a METHOD definition. It is a GLOBAL DS2 statement.
There is no ARRAY statement in DS2. Use the DECLARE or VARARRAY statement instead.

DS2 Arrays

An ARRAY assignment statement uses the := operator.

```
proc ds2;
data _null_;
   method init();
      dcl char(1) A[3] B[3];
      A:=('A','B','C');
      B:=A;
      put A[*]= B[*]=;
   end;
enddata;
run;
quit;
```

array-name:=(*constant-list*);
array-name:= *array-name*;

```
A[1]=A A[2]=B A[3]=C B[1]=A B[2]=B B[3]=C
```

ds03d08c

43

Setup for the Poll

Does the DS2 DATA program in the **ds03a01** program compile and execute without errors?

```
proc ds2;
data test/overwrite=yes;
   method init();
      vararray double n[5];
      n:=(1,2,3,4,5);
   end;
   method run();
      set orion.banks;
   end;
enddata;
run;
quit;
```

ds03a01

44

3.01 Poll

Does the DS2 DATA program in the **ds03a01** program compile and execute without errors?

- ○ Yes
- ○ No

45

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
COMMENT and * statements	/* */

```
data test1;
   comment Comment 1;
   *Comment 2;
   /*Comment 3*/
   set orion.continent;
   where Continent_ID=96;
   put _all_;
run;
```

```
proc ds2;
data test2 /overwrite=yes;
   /*Comment 1*/
   /*Comment 2*/
   /*Comment 3*/
   method run();
      set
      {select *
         from orion.continent
         where Continent_ID=96};
      put _all_;
   end;
enddata;
run;
quit;
```

ds03d09

48

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
DELETE	Subsetting IF

```
data test1;
   set orion.continent;
   if Continent_ID=96
      then delete;
run;
```

```
proc ds2;
data test2 /overwrite=yes;
   method run();
      set orion.continent;
      if Continent_ID ne 96;
   end;
enddata;
run;
quit;
```

ds03d11

50

DATA Steps versus DS2 DATA Programs

Many missing statements have equivalents in DS2.

Missing Statement	DS2 Equivalent
LINK	METHOD

```
data _null_;
   do i=1 to 3;
      link PrintIt;
   end;
return;
PrintIt:
   put 'PrintIt Routine: ' i=;
return;
run;
```

```
proc ds2;
data _null_;
   dcl int i;
   method PrintIt();
      put 'PrintIt Routine: ' i=;
   end;
   method run();
      do i=1 to 3;
         printit();
      end;
   end;
enddata;
run;
quit;
```

ds03d12

52

DS2: Not a DATA Step Replacement

Sometimes missing is really missing.

Missing Statement	Why?
FILE, INFILE, INPUT, LIST, LOSTCARD, CARDS, DATALINES	DS2 currently only reads from and writes to tables.
DISPLAY, DM, ABORT, ENDSAS, ERROR, DESCRIBE, MISSING	DS2 executes in a separate process, not in the SAS session.
EXECUTE, X	DS2 executes in a separate process, and cannot interact with the operating system.

55

Exercises

Level 1

4. Converting a DATA Step to a DS2 DATA Program

a. Open the **ds03e04** program. Convert the DATA step to a DS2 DATA program.

b. Convert the ATTRIB and LABEL statements to a DECLARE statement with a HAVING clause.

c. Place the executable statements (SET, assignment, and subsetting IF) in the RUN method.

d. Execute the modified program.

1) Review the SAS log to ensure that no errors or warnings were issued.

2) Review the output report. Ensure that it matches the original report.

Level 2

5. Converting a DATA Step to a DS2 DATA Program

a. Open the **ds03e05** program. Convert the DATA step to a DS2 DATA program.

b. Convert ARRAY statements to the appropriate DS2 statements.

c. Assign initial values to the **goal** array elements. Use a DS2 array assignment statement (:=).

d. Execute the modified program.

1) Review the SAS log to ensure that no errors or warnings were issued.

2) Review the output report. Ensure that it matches the original report.

Challenge

6. Converting to a DS2 DATA Program with User-Defined Methods

The **ds03e06** program contains a DATA step program that uses LINK statements to execute subroutines.

a. Convert this to a DS2 DATA program with user-defined methods that replace the subroutines.

b. Ensure that the DS2 DATA program produces the same results as the original, without producing warnings or errors in the SAS log.

3.3 Solutions

Solutions to Exercises

1. Reviewing an Undeclared Variable

a. Open the **ds03e01** program. Submit the program and review the SAS log. Notice the warning concerning the undeclared variable **Address**.

b. Add the SCOND=NONE option to the PROC DS2 statement. Submit the program and verify that no warnings or error messages are produced.

c. Modify the DATA _NULL_ statement to create the output data set **work.mailing**. Uncomment the PROC CONTENTS step at the end of the program. Submit the program and verify that no warnings or error messages are produced. Review the PROC CONTENTS output. What is the length of the variable **Address**? **200** bytes

d. Add a global DECLARE statement to the program, declaring the variable **address** as char(50). Remove the SCOND=NONE option from the PROC DS2 statement and add the /OVERWRITE=YES option to the DATA statement. Submit the program and review the SAS log. Verify that no warnings or error messages are produced. Then review the PROC CONTENTS output. What is the length of the variable **Address**? **50** bytes

```
/*ds03s01*/
/*a.*/
proc ds2;
data _null_;
   method run();
      set orion.supervisors;
      if upcase(country)='US' and City='San Diego';
      address=catx(',',catx(' ',street_number, street_name)
                      ,City, State);
      put address=;
   end;
enddata;
run;
quit;

/*b.*/
proc ds2 scond=none;
data _null_;
   method run();
      set orion.supervisors;
      if upcase(country)='US' and City='San Diego';
      address=catx(',',catx(' ',street_number, street_name)
                      ,City, State);
      put address=;
   end;
enddata;
run;
quit;
```

```
/*c.*/
proc ds2 scond=none;
data work.mailing;
   method run();
      set orion.supervisors;
      if upcase(country)='US' and City='San Diego';
      address=catx(',',catx(' ',street_number, street_name)
                     ,City, State);
      put address=;
   end;
enddata;
run;
quit;

proc contents data=work.mailing;
run;

/*d.*/
proc ds2;
data work.mailing/overwrite=yes;
   dcl char(50) address;
   method run();
      set orion.supervisors;
      if upcase(country)='US' and City='San Diego';
      address=catx(',',catx(' ',street_number, street_name)
                     ,City, State);
      put address=;
   end;
enddata;
run;
quit;

proc contents data=work.mailing;
run;
```

2. **Modifying a Program to Include an Undeclared Variable**

```
/*ds03s02*/
proc sql noprint;
select avg(salary)
   into :AvgSalary
   from orion.employee_payroll
;
quit;

proc ds2;
data higher / overwrite=yes;
   dcl double threshold;
   retain threshold;
   method init();
      threshold=2*&AvgSalary;
```

```
   end;
   method run();
      set orion.staff;
      if salary > threshold;
   end;
enddata;
run;
quit;

title "Salaries more than 2 times the average of
%sysfunc(putn(&AvgSalary,dollar10.2))";
proc print data=higher;
   var Employee_ID job_title salary threshold;
   format threshold dollar10.2;
run;
title;
```

3. Converting a DATA Step and Examining the Results

a. Open the **ds03e03** program. Submit the program.

b. How many observations are in the **transaction** data set that is used to produce the report titled "Cash or Check transactions"? **4**

c. Uncomment the second %LET statement and resubmit the program. How many observations are in the **transaction** data set that is used to produce the report titled "Credit Card transactions"? **27**

d. Convert the DATA step to a DS2 DATA program. Resubmit the program twice. Produce both the "Credit Card transactions" and "Cash or Check transactions" **transaction** data sets and associated reports. Verify that the modified program produces no errors or warnings in the SAS log.

```
/*ds03s03*/
/*****************************************
 TRANSACTION and SUM are DS2 reserved words
 Literal values require single quotation marks
*****************************************/
%let type=Cash or Check;
/*%let type=Credit Card;*/
proc ds2;
data "transaction"/overwrite=yes;
   dcl double "Sum" total;
   keep Employee_ID total "sum";
   method run();
      set orion.employee_donations;
      if paid_by=%tslit(&type)
         and not nmiss(qtr1,qtr2,qtr3,qtr4);
      "sum"=sum(qtr1,qtr2,qtr3,qtr4);
      total=sum("sum",.25*"sum");
   end;
enddata;
run;
quit;
```

```
title "&type transactions";
proc print data=transaction;
   var Employee_ID sum total;
run;
title;
```

4. Converting a DATA Step to a DS2 DATA Program

```
/*ds03s04*/
proc ds2;
data staffbonus /overwrite=yes;
   dcl double bonus having
       label 'Bonus Payment'
       format dollar10.2;
   keep Employee_ID salary bonus;
   method run();
      set orion.staff;
      bonus=salary*.05;
      if bonus > 10000;
   end;
enddata;
run;
quit;

title 'Bonuses > $10,000';
proc print data=staffbonus (obs=9);
   var Employee_ID salary bonus;
run;
title;
```

5. Converting a DATA Step to a DS2 DATA Program

```
/*ds03s05*/
proc ds2;
data funddrive/overwrite=yes;
   keep Employee_ID diff:;
   vararray double qtr[4];
   vararray double diff[4];
   dcl double goal[4];
   method init();
      goal:=(5,10,10,15);
   end;
   method run();
      dcl double i;
      set orion.employee donations;
      do i=1 to dim(qtr);
         diff[i]=sum(qtr[i],-goal[I]);
      end;
   end;
enddata;
run;
quit;

title "Fund Drive Performance";
proc sql;
select sum(diff1) 'Quarter 1'
```

```
      ,sum(diff2) 'Quarter 2'
      ,sum(diff3) 'Quarter 3'
      ,sum(diff4) 'Quarter 4'
   from funddrive
;
quit;
```

6. Converting to a DS2 DATA program with User-Defined Methods

```
/*ds03s06*/
proc ds2;
data funddrive (overwrite=yes);
   vararray double qtr[4];
   vararray double diff[4];
   dcl double goal[4];
   keep Employee_ID diff:;
   method calc(double q) returns double;
      return sum(qtr[q],-goal[q]);
   end;
   method init();
      goal:=(5,10,10,15);
   end;
   method run();
      dcl int i;
      set orion.employee_donations;
      do i=1 to 4;
         diff[i]=calc(i);
      end;
   end;
enddata;
run;
quit;

title "Fund Drive Performance";
proc sql;
select sum(diff1) 'Quarter 1'
      ,sum(diff2) 'Quarter 2'
      ,sum(diff3) 'Quarter 3'
      ,sum(diff4) 'Quarter 4'
   from funddrive
;
quit;
```

Solutions to Student Activities (Polls/Quizzes)

3.01 Poll – Correct Answer

Does the DS2 DATA program in the **ds03a01** program compile and execute without errors?

- ○ Yes
- ◉ No

VARARRAY is a global statement, so it is not valid inside a METHOD definition. The VARARRAY statement must be relocated outside of the METHOD definitions so that the code can compile and run properly.

46

Chapter 4 New Data Types and Syntax

4.1 DATA Program Structuring

Objectives

- Describe the use and placement of PROC DS2 global statements.
- Describe the required statement and code block sequence within a DS2 DATA program.
- Identify the conditions requiring the use of the FORWARD statement.

3

PROC DS2 Global Statements

- DS2 global statements are valid only outside of program blocks
- DS2 global statements include
  - DROP PACKAGE
  - DROP THREAD
  - DS2_OPTIONS

4

The DROP PACKAGE Statement

- Must be used outside of any other program block.
- Deletes an existing DS2 package.

```
proc ds2;
package test;
method t();
   put 'Testing again';
end;
endpackage;
run;
data _null_;
   dcl package test tst();
   method init();
     tst.t();
   end;
enddata;
run;
drop package test;
run;quit;
```

DROP PACKAGE *package_name;*

ds04d01e

5

The DROP THREAD Statement

- Must be used outside of any other program block.
- Deletes an existing DS2 thread.

```
proc ds2;
thread mythread;
   method run();
      set orion.banks;
      put _all_;
   end;
endthread;
run;
data _null_;
   dcl thread mythread th();
   method run();
      set from th threads=4;
   end;
enddata;
run;
drop thread mythread;
run;
quit;
```

DROP THREAD *thread_name;*

ds04d01e

6

The DS2_OPTIONS Statement

- Specifies or changes the default behavior of a DS2 DATA, PACKAGE or THREAD program.
- Must appear immediately before a DS2 DATA, PACKAGE or THREAD statement.
- Applies only to the next DATA, PACKAGE, or THREAD statement.

```
DS2_OPTIONS option(s);
```

7

The DS2_OPTIONS Statement

Options useful for controlling program execution:

Option	Effect	Default
DIVBYZERO= ERROR | IGNORE	IGNORE: writes missing or null value to the result set, no message in the SAS log.	ERROR: halts row processing, writes an error to the SAS log.
SAS	Overrides the ANSIMODE system or DS2 procedure option	N/A. Valid only when in ANSIMODE
SCOND= WARNING|NONE| NOTE|ERROR	specifies the level of message issues for undeclared variables	WARNING

8

The DS2_OPTIONS Statement

Options useful for debugging:

Option	Effect	Default
MISSING_NOTE	NOTE in the SAS log if invalid function argument generates missing value	ERROR in the SAS log if invalid function argument generates missing value
TRACE	Tracing of statements executed in-database. Useful for debugging.	N/A
TYPEWARN	Prints a warning in the SAS log for every implicit data type conversion	N/A

9

Block Global Declaration Statements

- Block global declaration statements
  - Are placed in a program block before any method definitions
  - Are not valid in the context of a DS2 method.
- Examples of global declaration statements include:
  - RETAIN
  - DROP / KEEP
  - DECLARE (for global variables)
  - FORWARD
  - RENAME
  - VARARRAY

10

The FORWARD Statement

- Indicates that the method definition follows the first use of the method in an expression.

```
proc ds2;
data _null_;
   dcl double Tf Tc;
   forward c2f f2c;
   method run();
      Tf=c2f(37);
      Tc=f2c(Tf);
      put Tc= Tf=;
   end;
   method c2f(double Tc) returns double;
      return((Tc*9/5)+32);
   end;
   method f2c(double Tf) returns double;
      return((Tf-32)*5/9);
   end;
enddata;
run;
quit;
```

FORWARD *method_name <...method_name;*

ds04d01e

11

4.2 Data Types

Objectives

- Identify DS2 data types corresponding to Base SAS numeric and character types.
- Describe the new data types available in DS2.
- Explain how new DS2 data types are handled when they are stored in SAS data sets.
- Describe the consequences of automatic type conversion on numeric precision.

13

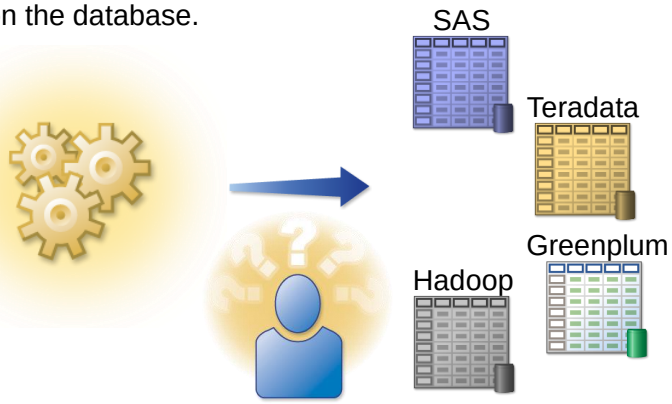
Business Scenario

Orion Star programmers want to leverage the new data types that are available in DS2 to conduct precise calculations on large numbers and to optimize data storage.

14

DS2 Data Types

- DS2 can define and manipulate many ANSI data types.
- Types that are available for storage depend on the database.



15

Character Types

DS2 Character Data Types

Variable Type	Description
CHAR(*n*)	Fixed-length character
NCHAR(*n*)	Fixed-length Unicode character
VARCHAR(*n*)	Variable-length character data
NVARCHAR(*n*)	Variable-length Unicode character data

- *n* is the maximum number of characters.
- CHAR and VARCHAR use one byte per character.
- Unicode characters use two or four bytes per character.

CHAR is the equivalent of a SAS character variable. It is the default type for undeclared character values.

16

Fractional Numeric Data Types

In Base SAS, all numeric values are fractional.

- signed, approximate, double-precision floating point
- maximum of eight bytes
- maximum of 16- or 17-digit precision

```
data _null_;
   a=123456789123456789234.56789;
   b=9876543219876543219876.54321;
   c=a+b+.00001;
   put '***** SAS NUMERIC *****';
   put ' 123456789123456789234.56789 / ' a best29.5;
   put ' 9876543219876543219876.54321 / ' b best29.5;
   PUT '111111111111111111111111.11111 / ' c best29.5;
run;
```

```
***** SAS NUMERIC *****
 123456789123456789234.56789 /  123456789123456801776
 9876543219876543219876.54321 /  987654321987654569164
111111111111111111111111.11111 / 111111111111111113441280
```

ds04d01a

17

Because of differences in the hardware architecture and operating systems, under optimal conditions, the maximum number of significant digits that can be represented in eight bytes is 17 on IBM mainframes, but only 16 on UNIX and Windows systems. See ***SAS® 9.4 Language Reference: Concepts – SAS Variables – Numerical Accuracy in SAS*** for more information on numerical accuracy, precision vs. magnitude, and floating-point representation of numbers.

Fractional Numeric Data Types

DS2 Fractional Numeric Data Types

Variable Type	Description
DECIMAL(*p*,*s*)	Signed, exact, fixed-point decimal of user-defined precision (all digits - max of 52) and scale (decimal digits)
DOUBLE	Signed, approximate, double-precision floating point
FLOAT(*p*)	Signed, approximate, floating point decimal of user-defined precision. Precision determines whether it is stored as DOUBLE or REAL.
REAL	Signed, approximate, single-precision floating point

DOUBLE is analogous to a DATA step numeric.
It is the default type for undeclared numeric values.

18

In DS2, NUMERIC is an alias for DECIMAL. In the following example, the **MyDec** and **MyNum** variables are equivalent:

```
dcl decimal(40,5) MyDec;
dcl numeric(40,5) MyNum;
```

Fractional Numeric Data Types

DS2 DOUBLE is analogous to a SAS numeric.

```
proc ds2;
data _null_;
method init();
   dcl double a b c;
   a=1234567891234567891234.56789;
   b=9876543219876543219876.54321;
   c=a+b+.00001;
   put '***** DOUBLE TYPE *****';
   put ' 1234567891234567891234.56789 / ' a best29.5;
   put ' 9876543219876543219876.54321 / ' b best29.5;
   put '111111111111111111111111.11111 / ' c best29.5;
end;
enddata;
run;
quit;
```

```
***** DOUBLE TYPE *****
 1234567891234567891234.56789 /  12345678912345680117776
 9876543219876543219876.54321 /  98765432198765435944960
111111111111111111111111.11111 / 111111111111111111344128
```

ds04d01b

19

Setup for the Poll

In this program, variables **a**, **b**, and **c** are declared as DECIMAL instead of DOUBLE.

```
proc ds2;
data _null_;
method init();
   dcl decimal(30,5) a b c;
   a=1234567891234567891234.56789;
   b=9876543219876543219876.54321;
   c=a+b+.00001;
   put '***** DECIMAL TYPE *****';
   put ' 1234567891234567891234.56789 / ' a best29.5;
   put ' 9876543219876543219876.54321 / ' b best29.5;
   put '11111111111111111111111.11111 / ' c best29.5;
end;
enddata;
run;
quit;
```

DECIMAL(30,5) - 30 significant digits, five of which are fractional

ds04a01

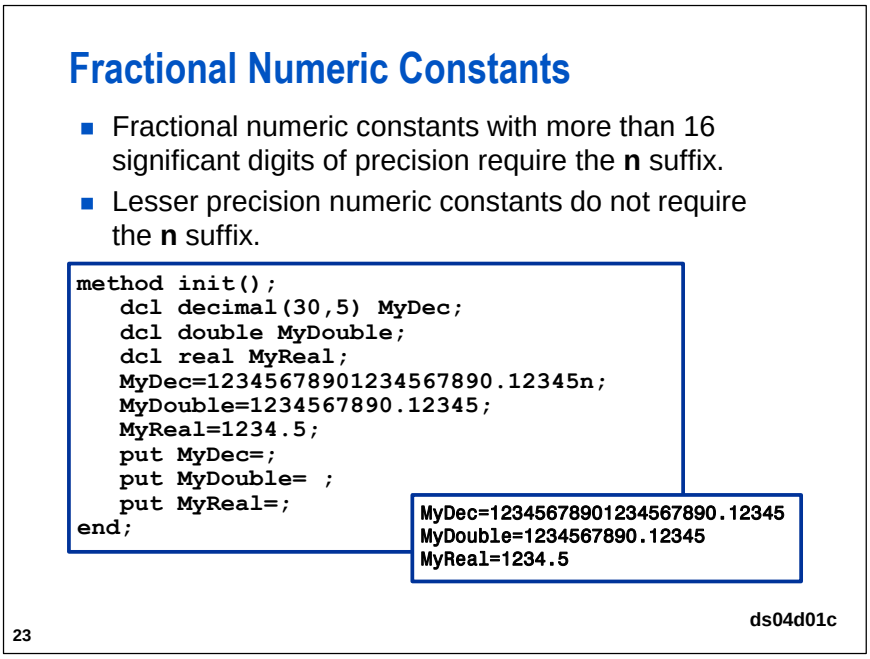
20

4.01 Poll

Submit the program **ds04a01** and review the SAS log. Did the use of the DECIMAL data type produce the expected precision in the results?

- ○ Yes
- ○ No

21



The default data type for all numeric values in DS2 is DOUBLE. The **n** suffix declares a numeric constant using the higher-precision NUMERIC data type.

Storage Location and Data Type

This demonstration illustrates that, although DS2 can manipulate many ANSI data types, the storage mechanism still determines which data types can be saved to or read from tables.

1. Open the **ds04d01d** program.
2. Examine the section labeled "Part A: Pure data manipulation." Notice the following:
   a. The DS2 DATA program declares
      - three numeric variables, one each of type DECIMAL, DOUBLE, and REAL
      - three text variables to hold the equivalent values as text.

      1) The program assigns values to all the variables.

      2) The values are then written to the SAS log and the program terminates.

   b. Submit Part A and examine the SAS log. Notice that the printed numeric values match the text variable values. This proves that the DECIMAL and REAL variable values retained full precision during processing.
3. Examine the section labeled "Part B: Storing values in Base SAS data sets." Notice the following:
   a. The DS2 DATA program declares
      - three numeric variables, one each of type DECIMAL, DOUBLE, and REAL
      - three text variables to hold the equivalent values as text.

      1) The program assigns values to all the variables.

      2) The values are then written to the SAS log before the program terminates.

      3) The DATA program produces a SAS data set named **work.test** with one observation.

   b. Submit Part B. Examine the SAS log. Notice that the printed numeric values match the text variable values. This proves that the DECIMAL and REAL variable values retained full precision during processing.
4. Examine the section labeled "Part C: DECIMAL and REAL values stored in a Base SAS data set." Notice the following:
   a. The PROC SQL selects all variables from the **work.test** data set that was produced by the previous DS2 DATA program.
   b. Submit Part C. Examine the SAS log.

      1) Notice that the PROC SQL step executes without errors or warnings.

      2) Examine the PROC SQL output. Notice that the DECIMAL numeric value was truncated due to a loss of precision when the value was converted to double precision for storing the Base SAS data set.

Integer Numeric Data Types

DS2 Integer Numeric Data Types

Variable Type	Description
BIGINT	Signed, exact whole numbers, 19 digit max
INTEGER	Signed, exact whole numbers, 10 digit max
SMALLINT	Signed, exact whole numbers, 5 digit max
TINYINT	One byte, signed, whole numbers between –128 and 127

BIGINT values might also be truncated when you store data in locations that do not support high-precision numerics.

26

Integer Numeric Constants

- Integer numeric constants do not require the **n** suffix.

```
method init();
   dcl bigint   MyBig;
   dcl integer  MyInt;
   dcl smallint MySmall;
   dcl tinyint  MyTiny;
   MyBig  =1234567890123456789;
   MyInt  =1234567890;
   MySmall=12345;
   MyTiny=127;
   put MyBig=;
   put MyInt=;
   put MySmall=;
   put MyTiny=;
end;
```

```
MyBig=1234567890123456789
MyInt=1234567890
MySmall=12345
MyTiny=127
```

Integer constants are limited to 19 significant digits.

ds04d02

27

Setup for the Poll

Submit the **ds04a02** program and review the SAS log.
Why does the program not produce the correct answer?

```
proc ds2;
data _null_;
method init();
   dcl decimal(5) Dec Ans;
   Dec=35;
   Ans=dec/8;
   /*Correct answer:4.375*/
   put Dec= Ans=;
end;
enddata;
run;
quit;
```

28

4.03 Multiple Choice Poll

Submit the **ds04a02** program and review the SAS log.
Why does the program not produce the correct answer?

a. The formula used to calculate the result is incorrect.
b. The results are formatted incorrectly.
c. The decimal variable **Ans** does not accommodate fractional values.
d. The numeric constant should use the **n** suffix.

29

Binary Data Types

DS2 supports two binary data types.

Variable Type	Description
BINARY(*n*)	Fixed-length binary data, where *n* is the number of bytes used to store all values.
VARBINARY(*n*)	Variable-length binary data, where *n* is the maximum number of bytes. Only the number of bytes required to represent the actual value is used for storage.

31

Date and Time Data Types

DS2 supports ANSI date and time data types.

Variable Type	Description
DATE	Calendar date
TIME	Clock time
TIMESTAMP	Datetime value

32

Date and Time Constants

- Constants are prefixed with a keyword (date, time, timestamp).
- Values are enclosed in single quotation marks.
- Dates use the yyyy-mm-dd format.
- Times use the hh:mm:ss<.ddddd> format.

```
method init();
   dcl date MyDate;
   dcl time MyTime;
   dcl timestamp MyTimestamp;
   MyDate=date'2013-09-07';
   MyTime=time'17:00:00';
   MyTimestamp=timestamp '2013-09-07 17:00:00';
   put MyDate= MyTime=;
   put MyTimestamp=;
end;
```

```
MyDate=2013-09-07 MyTime=17:00:00
MyTimestamp=2013-09-07 17:00:00
```

ds04d03

33

4.04 Quiz

Execute the **ds04a03** program and review the SAS log.

a. In the SAS data set **work.test**, what type is the variable **DesiredSalary**?

b. In the Teradata table **my_db.test**, what type is the variable **DesiredSalary**?

34

4.3 Automatic Data Type Conversion

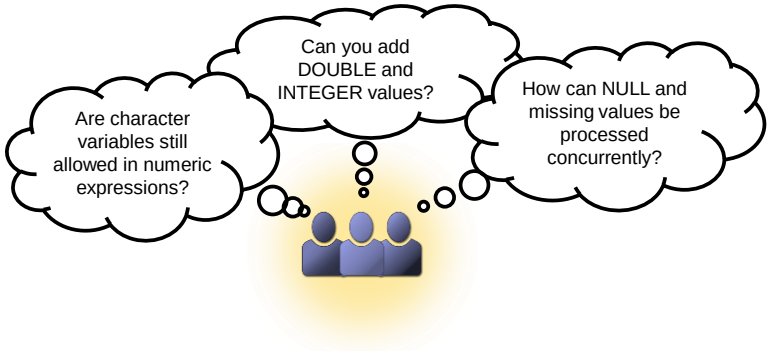
Objectives

- Describe when automatic conversion of data types occurs.
- List coercible and non-coercible data types.
- Contrast the handling of SAS MISSING and ANSI NULL values in PROC DS2 SAS mode vs. ANSI mode.

38

Business Scenario

Orion Star programmers have years of DATA step experience where only two types of variables exist. They have many questions about the implications of processing with multiple new data types.



39

Automatic Type Conversions

- Expression operands must be the same data type for the expression to resolve.
- A data type can be coercible or non-coercible.
  - Non-coercible data types automatically convert only to character data types.
  - Coercible data types automatically convert to multiple data types.

40

Automatic Type Conversions

- Character and numeric data types are coercible.
  - DECIMAL, DOUBLE, REAL
  - BIGINT, INTEGER, SMALLINT, TINYINT
  - CHAR, NCHAR, VARCHAR, NVARCHAR
- Date, time, and binary data types are not coercible.
  - DATE, TIME, TIMESTAMP
  - BINARY, VARBINARY

41

Automatic Type Conversions

Standard Numeric Conversion

- Numeric values are promoted to the highest precedence type.
  - DOUBLE is the highest numeric precedence.
- Examples:
  - SMALLINT to INTEGER
  - REAL to DOUBLE
  - CHAR to DOUBLE

42

See the documentation topic *DS2 Type Conversion* for more details.

Automatic Type Conversions

Standard Character Conversion

- Character values are promoted to the highest precedence type.
  - CHAR is the highest character precedence.
- Examples:
  - NCHAR to CHAR
  - DECIMAL to CHAR

43

Automatic Type Conversions

Automatic conversions occur when

- character types are used in numeric expressions
- numeric types are used in character expressions
- method calls provide parameter values not matching parameter data types defined in the method
- values are used in logical, arithmetic, relational, and concatenation expressions.

DATA step programmers must remain alert to the consequences of processing more than two data types.

44

Automatic Data Type Conversion

1. Open the **ds04d08a** program.
2. Submit the portion marked "Part A: Standard Numeric Conversions." Review the SAS log and observe the following results:
   a. Three variables are declared, one each of type REAL, DOUBLE, and INTEGER.
   b. Numeric constants were assigned to the three numeric variables. The results printed in the SAS log are exactly as expected.
   c. A smaller value was assigned to **MyReal**. When the variable is written to the SAS log, the value is not exact. Decimal values must be converted to binary for processing and storage. REAL data types provide only four bytes for numeric value storage. Because the decimal value 125.1 cannot be expressed exactly as a binary number, an approximation is required. This constraint applies to all uses of REAL data types.
   d. A value with 16 significant digits was assigned to **MyDouble**. When the variable is written to the SAS log, the value is not exact. Again, the decimal-to-binary conversion is the culprit, because this value cannot be expressed exactly as a binary number using 64 bits (8 bytes). ***See SAS® 9.4 Language Reference: Concepts – SAS Variables – Numerical Accuracy in SAS*** for more information on numerical accuracy, precision vs. magnitude, and floating-point representation of numbers.
   e. Subsequent mathematical operations using mixed numeric data types produce expected results. As expected, attempting to store a fractional numeric result in an INTEGER data type results in truncation of the information to the right of the decimal point.

Partial SAS Log

```
*** Numeric Constant Assignment Results ***
Code:    MyReal=125.125 MyDouble=100000000.00001 MyInt=1000
Result:  MyReal=125.125 MyDouble=100000000.00001 MyInt=1000
```

```
*** High Precision Numeric Constant Assignment Results ***
Code:    MyReal=125.125n MyDouble=100000000.00001n MyInt=1000n
Result:  MyReal=125.125  MyDouble=100000000.00001  MyInt=1000

*** 32-bit numerics - limitations of REAL values ***
Code:    MyReal=125.1
Result:  MyReal=125.099998474121

*** 125.1 cannot be exactly represented in 4 bytes    ***
*** (32 bits), so value is approximate.               ***

*** 64-bit numerics - limitations of DOUBLE values ***
Code:    MyDouble=9007199254740993
Result:  MyDouble=9007199254740992

*** 9007199254740993 cannot be exactly represented in  ***
*** 8 bytes/64 bits, so value is approximate.          ***
*** See http://support.sas.com/techsup/technote/ts654.pdf ***

*** Calculations with diverse numeric types ***
Code:   MyDouble=MyInt+MyReal***
Result: MyDouble=1125.125 MyInt=1000 MyReal=125.125

Code:   MyReal=MyInt+MyDouble
Result: MyReal=1100.125 MyInt=1000 MyDouble=100.125

Code:   MyInt=MyReal+MyDouble***
Result: MyInt=225 MyReal=125.125 MyDouble=100.125
```

3. Clear the SAS log.
4. Submit the portion marked “Part B: Standard Character Conversions.” Review the SAS log and observe the following results:
   a. Standard character constants were assigned to the CHAR and VARCHAR variables. Unicode character constants (using the N prefix) were assigned to the NCHAR and NVARCHAR variables.
   b. When the variable values are printed in the SAS log, they are exactly as expected. Notice that in this demonstration, character variable values are printed to the SAS log with an asterisk (*) prepended and appended to make leading and trailing blanks visible. Fixed-width (CHAR and NCHAR) values are padded with spaces. Variable-width variable values (VARCHAR and NVARCHAR) are not padded.
   c. Next, standard character constants were stored in the NCHAR and NVARCHAR variables and Unicode character constants were stored in the CHAR and VARCHAR variables, which requires automatic type conversion.
   d. When the variable values are printed in the SAS log, they also are exactly as expected.
   e. Concatenations of variable-width values do not have trailing spaces padded on the end of each value in the concatenation results.
   f. Concatenations of fixed-width values include the trailing spaces padded on the end of each value in the concatenation results. In fact, the concatenation results were longer than the space available when assigned to MyVARCHAR, so the value was truncated.

Partial SAS Log

```
*** In the following section, character variable values  ***
*** displayed with leading and trailing bars (*) to show ***
*** leading and trailing blanks in the values            ***

*** Text and Unicode Text Constant Assignment Statements ***
Code: MyCHAR='Pants' MyNCHAR=n'Calça'
MyCHAR=* Pants                                * MyNCHAR=* Calça                              *
Code: MyVARCHAR='Jacket and slacks' MyNVARCHAR=n'Paletó e calça'
MyVARCHAR=* Jacket and slacks * MyNVARCHAR=* Paletó e calça *

*** Cross-type assignments***
Code: MyNCHAR='Pants' MyCHAR=n'Calça'
MyNCHAR=* Pants                                * MyCHAR=* Calça                               *

Code: MyNVARCHAR='Jacket and slacks' MyVARCHAR=n'Paletó e calça'
MyNVARCHAR=* Jacket and slacks * MyVARCHAR=* Paletó e calça *

*** Concatenation: VARCHAR and NVARCHAR into CHAR***
Code: MyCHAR='*'||MyVARCHAR||'*'||MyNVARCHAR||'*'
MyCHAR=*Jacket and slacks*Paletó e calça*

Code: MyCHAR=cat('*',CATX('*',MyVARCHAR,MyNVARCHAR),'*'
MyCHAR=*Jacket and slacks*Paletó e calça*

*** Concatenation: CHAR and NCHAR into VARCHAR***
Code: MyVARCHAR='*'||MyCHAR||'*'||MyNCHAR||'*'
MyVARCHAR=*Pants                               *Calça
*** result is truncated (missing the *) due to ***
*** trailing spaces in the variable values     ***

Code: MyVARCHAR=cat('*',CATX('*',MyCHAR,MyNCHAR),'*'
MyVARCHAR=*Pants*Calça
```

Idea Exchange

Submit the **ds04a04** program and review the SAS log. What differences do you notice from DATA step, automatic, character-to-numeric and numeric-to-character conversions?

46

Missing and Null Values

NULL and MISSING are processed very differently.

- ANSI NULL is an unknown value.
- SAS MISSING is a known value.

These values are most significant when

- filtering data
- joining data

47

DS2 Processing Modes

DS2 has two processing modes.

- SAS mode (default)
  - Unknown intermediate results are represented as missing values.
  - A blank constant (' ') is interpreted as MISSING.
  - In SAS mode, NULL auto-converts to MISSING under the following conditions:
    - When used in computations with or assigned to DOUBLE or CHAR() variables.
    - When passed to a Base SAS function.

48

continued...

DS2 Processing Modes

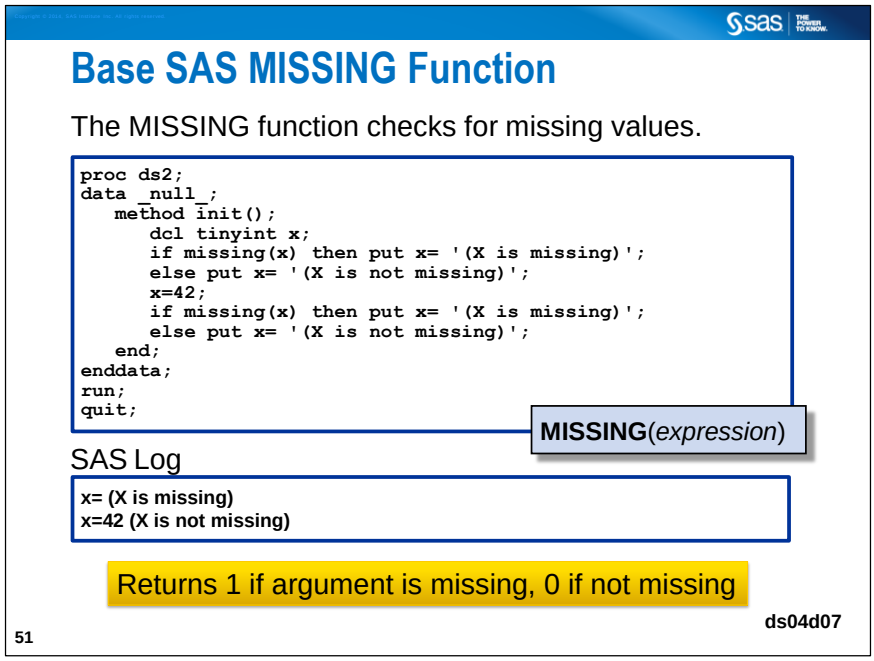
- ANSI mode
  - ANSI mode is invoked with the ANSIMODE option.
  - Unknown intermediate results are represented by NULL values.
  - A blank constant (' ') is interpreted as a space character.
  - NULL values passed to base SAS functions are still auto-converted to MISSING.
  - The ANSI mode auto-converts numeric MISSING values to NULL, but not character missing values.
  - Missing character variables read from SAS data sets appear as blank-filled character strings and are evaluated as not NULL by the NULL function and not MISSING by the MISSING function.

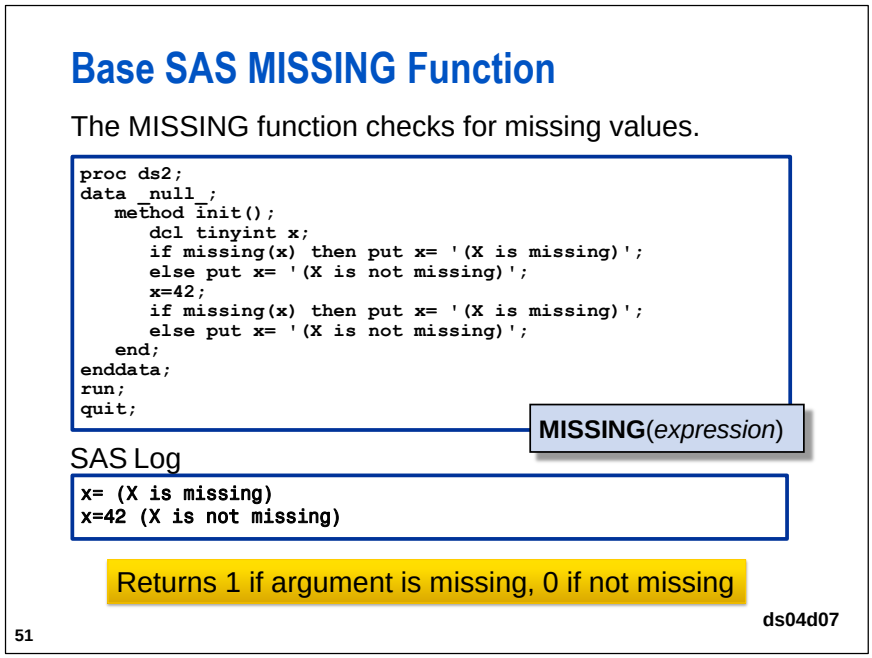
49

DS2 Processing Modes

- Null and missing values can be processed together in either mode.
- Special SAS numeric missing values (.A-._) are lost in the following situations:
  - in ANSI mode
  - when you perform MISSING to NULL to MISSING conversions in SAS mode

50





Processing Null and Missing Values in DS2

1. Open the **ds04d08b** program and examine the section labeled "Part A.1: Numeric Data SAS Mode Evaluation" Notice the following:

   a. The variable **Dec** is a DECIMAL data type. It does not accept SAS missing values, but uses NULL instead. **Dec** is null.

   b. The variable **Num** is a DOUBLE data type (standard SAS numeric). **Num** accepts SAS missing values. **Num** is missing.

   c. The value in **Dec** is evaluated for equality to the SAS missing numeric constant (.) and also evaluated by the MISSING and NULL functions.

   d. The value in **Num** is evaluated for equality to the SAS missing numeric constant (.) and also evaluated by the MISSING and NULL functions.

   e. Submit Part A.1 and review the SAS log. Expected results indicate that the NULL value in **Dec** was automatically converted to MISSING for evaluation, but the MISSING value in **Num** was not automatically converted to NULL.

   SAS Log

```
SAS Mode ****************************************
 - Variable Dec is null
 - Variable Num is missing
 - Nonexistent data values are all treated as SAS missing values.

Is Dec evaluated as missing in SAS mode?
   Dec=.
   Dec evaluates as both MISSING and NULL

Is Num evaluated as NULL in SAS mode?
   Num=.
   Num evaluates as MISSING but not NULL ( Num=. )
```

2. Examine the next section of the **ds04d08b** program labeled "Part A.2: ANSI Mode Evaluation." Notice the following:

   a. The variable **Dec** is a DECIMAL data type. It does not accept SAS missing values, but uses NULL instead. **Dec** is null.

   b. The variable **Num** is a DOUBLE data type (standard SAS numeric). **Num** accepts SAS missing values. **Num** is missing.

   c. The value in **Dec** is evaluated for equality to the SAS missing numeric constant (.) and also evaluated by the MISSING and NULL functions.

   d. The value in **Num** is evaluated for equality to the SAS missing numeric constant (.) and also evaluated by the MISSING and NULL functions.

   e. Submit Part A.2 and review the SAS log. In ANSI mode, the NULL value in **Dec** was not converted to MISSING for evaluation against the missing constant (.), so **Dec** was ***not equal*** to the missing constant. However, both the NULL and MISSING functions evaluated **Dec** as true. The MISSING value in **Num** automatically converted to NULL for evaluation, so **Num** was also ***not equal*** to the missing constant. Both the NULL and MISSING functions that were applied to **Num** produced true results.

SAS Log

```
ANSI Mode ****************************************
 - Variable Dec is null
 - Variable Num is missing
 - MISSING values convert to NULL at input or assignment.

Is Dec evaluated as missing in ANSI mode?
   Dec is NOT =.  ( Dec= )
   Dec evaluates as both MISSING and NULL

Is Num evaluated as NULL in ANSI mode?
   Num is NOT =. ( Num= )
   Num evaluates as both MISSING and NULL
```

3. Examine the next section of the **ds04d08b** program labeled "B.1: SAS Mode Numeric Data - Special Missing Values." Notice the following:

   a. The variable **Dec** is a DECIMAL data type. It does not accept SAS missing values, but uses NULL instead.

   b. The variable **Num** is a DOUBLE data type (standard SAS numeric), which requires SAS missing values instead of NULL.

   c. **Num** is assigned *.A* as the special SAS missing value.

   d. The value in **Num** is evaluated for equality to SAS missing numeric constants (. and *.A*) and also evaluated by the MISSING and NULL functions.

   e. **Num** is copied to **Dec**, and then back to **Num**. This forces conversion to NULL, and then back to MISSING.

   f. The value in **Num** is reevaluated for equality to SAS missing numeric constants (. and *.A*) and reevaluated by the MISSING and NULL functions. The special missing value *.A* is lost in the conversion.

   SAS Log

```
SAS Mode *****************************************
 - Variable Num is missing (special missing value .A)
 - Nonexistent data values are all treated as SAS missing values.

Value before conversion:
   Num=A

Before conversion, is Num still evaluated as MISSING and not NULL in SAS mode?
   Num=.A
   Num evaluates as MISSING but not NULL ( Num=A )

Value after conversion:
   Num=.
   Special missing value .A has become a default missing (.) value

After conversion, is Num still evaluated as MISSING and not NULL in SAS mode?
   Num=.
   Num evaluates as MISSING but not NULL ( Num=. )
SAS Mode *****************************************
Variable Dec was not assigned a value - it is null
Variable Num contains the special missing value .A
Values before the conversion:
Dec= Num=A _N_=1
```

```
Value in Num was converted to NULL then back to missing.
Values after the conversion:
Dec= Num=. _N_=1

Special missing value was lost.

But missing values still handled as expected:
NULL -> MISSING is automatic for evaluation
Yes, Dec contains a missing value.
Yes, Dec is NULL.

MISSING -> NULL not automatic
Yes, Num contains a missing value.
No, Num is not NULL
```

4. Examine the section of **ds04d08b** labeled "Part B.2: ANSI Mode – Special Numeric Missing Values." Notice the following:

   a. The variable **Dec** is a DECIMAL data type. It does not accept SAS missing values, but uses NULL instead.

   b. The variable **Num** is a DOUBLE data type (standard SAS numeric), which requires SAS missing values instead of NULL.

   c. **Num** is assigned *.A* as the special SAS missing value.

   d. The value in **Num** is evaluated for equality to SAS missing numeric constants (. and *.A*). ANSI mode converts MISSING values to NULL for evaluation, so you expect that the compared value is equal to neither.

   e. **Num** is also evaluated by the MISSING and NULL functions.

   f. **Num** is copied to **Dec**, and then back to **Num** to force conversion to NULL and back to MISSING.

   g. The value in **Num** is then re-evaluated, testing for equality to SAS missing numeric constants (. and *.A*) and using the MISSING and NULL functions.

   SAS Log

```
ANSI Mode ****************************************
 - Variable Dec is null
 - Variable Num is missing (special missing value .A)
 - MISSING values convert to NULL at input or assignment.

Value before conversion:
   Num=
   Special missing value .A is converted to NULL upon evaluation

Before conversion, is Num still evaluated as MISSING and not NULL in ANSI mode?
   Num is NOT =. or .A ( Num= )
   Num evaluates as both MISSING and NULL

Value after conversion:
   Num=
   Missing value . is also converted to NULL upon evaluation

After conversion, is Num still evaluated as MISSING and not NULL in ANSI mode?
   Num is NOT =. or .A ( Num= )
   Num evaluates as both MISSING and NULL
```

5. In **ds04d08b** examine the section labeled “Part C.1: ANSI Mode Character Data Reading from SAS data set” Notice the following:

   a. In the data set **work.test**, the variable **C** contains standard SAS character data type, and accepts SAS character missing values. The first and last rows of the data set have values for **C**, but in the second observation **C** is missing. This data set is read into two DS2 DATA programs where the value of C is evaluated using the NULL and MISSING functions as well as the “ ” constant. One DATA program runs in SAS mode, the other in ANSI mode.

b. Submit Part C.1 and review the SAS log.

   SAS Log

```
****************************
Reading SAS Data in SAS Mode

C is not NULL -  c=Mark
C is not MISSING -  c=Mark
C is not - " " -  c=Mark
C is not NULL -  c=
C is MISSING
C =" "
****************************
┬ more SAS log ┬
****************************
Reading SAS Data in ANSI Mode

C is not NULL -  c=Mark
C is not MISSING -  c=Mark
C is not - " " -  c=Mark
C is not NULL -  c=
C is not MISSING -  c=
C =" "
****************************
```

6. In **ds04d08b** examine the section labeled “Part C.2: SAS Mode Expression Evaluation.” Notice the following:

   a. The variable **Ch** is a CHAR (standard SAS character) data type, and accepts SAS character missing values.

   b. The variable **VCh** is a VARCHAR data type, and does not accept a SAS character missing value. It requires a NULL value instead.

   c. The value in **Ch** is evaluated for equality to the SAS missing character constant (‘ ’) and also evaluated by the MISSING and NULL functions.

   d. The value in **VCh** is evaluated for equality to the SAS missing character constant (‘ ’) and also evaluated by the MISSING and NULL functions.

   e. Submit Part A.1 and review the SAS log. Expected results indicate that the NULL value in **VCh** was automatically converted to MISSING for evaluation, but the MISSING value in **Ch** was not automatically converted to NULL.

SAS Log

```
SAS Mode MISSING/NULL Evaluation *************************
 - Variable VCh is null
 - Variable Ch contains a SAS missing value
 - Nonexistent data values are all treated as SAS missing values
 - In SAS mode," " represents MISSING

Is Ch evaluated as missing?
   Ch =" "
   Ch is MISSING but not NULL.

Is VCh evaluated as missing in SAS mode?
   VCh =" "
   VCh is MISSING but not NULL.
```

7. Examine the next section of the **ds04d08b** program labeled "Part C.3: ANSI Mode Expression Evaluation." Notice the following:

   a. The variable **Ch** is a CHAR (standard SAS character) data type, and accepts SAS character missing values.

   b. The variable **VCh** is a VARCHAR data type, and does not accept a SAS character missing value. It requires a NULL value instead.

   c. The value in **Ch** is evaluated for equality ' ' and is also evaluated by the MISSING and NULL functions. (In ANSI mode, this represents a space character.)

   d. The value in **VCh** is evaluated for equality to ' ' and also evaluated by the MISSING and NULL functions.

   e. Submit Part A.2 and review the SAS log. Expected results indicate that neither **Ch** nor **VCh** is equal to a space character. Both the NULL and MISSING functions evaluate to *TRUE* when applied to either **Ch** or **VCh**.

SAS Log

```
ANSI Mode MISSING/NULL Evaluation ************************
 - Variable VCh is null
 - Variable Ch contains a SAS missing value
 - MISSING values convert to NULL at input or assignment.
 - " " represents a SPACE instead of a MISSING value.

Is Ch evaluated as missing?
   Ch is NOT =" "
   Ch is both MISSING and NULL.

Is VCh evaluated as missing in SAS mode?
   VCh is NOT =" "
   VCh is both MISSING and NULL.
```

It is possible to process both MISSING and NULL values in DS2 in either SAS or ANSI mode, as long as the default behavior demonstrated above is understood.

Idea Exchange

Submit the **ds04a05** program and review the SAS log.

Discuss the differences seen in the results when you concatenate character variables that contain missing values in DS2 SAS mode versus ANSI mode.

54

4.4 Expressions

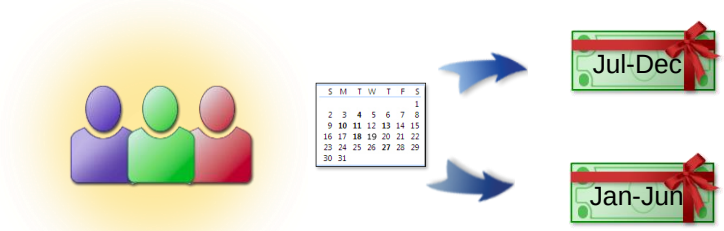
Objectives

- Perform conditional processing in the DS2 DATA program.
- Compare DS2 IF and SELECT expressions to the DATA step function IFN().
- Introduce the LIKE expression

57

Business Scenario

The Orion Star charity campaign manager wants to know whether employees tend to contribute more in the second half of the year. Propensity to donate can be expressed as the ratio of donor giving in one half of the year to donor giving in the other half of the year.



58

Calculating Propensity to Donate

DATA Step

```
data work.ratio;
   format Prop1 Prop2 6.2;
   set orion.employee_donations(drop=Recipients Paid_By);
   FirstHalf=sum(Qtr1,Qtr2,0);
   SecondHalf=sum(Qtr3,Qtr4,0);
   Prop1=FirstHalf/SecondHalf;
   Prop2=SecondHalf/FirstHalf;
run;
```

ds04d05

59

Calculating Propensity to Donate

Partial Log

```
NOTE: Division by zero detected at line 1414 column 20.
Prop1=0.00 Prop2=. Employee_ID=120265 Qtr1=. Qtr2=. Qtr3=. Qtr4=25
 FirstHalf=0 SecondHalf=25
_ERROR_=1 _N_=1

NOTE: Division by zero detected at line 1414 column 20.
Prop1=0.00 Prop2=. Employee_ID=120663 Qtr1=. Qtr2=. Qtr3=5 Qtr4=.
 FirstHalf=0 SecondHalf=5
_ERROR_=1 _N_=10
```

Data errors are from division by zero.

ds04d05

60

Calculating Propensity to Donate

Partial Results

Donation Propensity
Base SAS DATA Step

Obs	Employee_ID	FirstHalf	SecondHalf	Prop1	Prop2
1	120265	0	25	0.00	
2	120267	30	30	1.00	1.00
3	120269	40	40	1.00	1.00
4	120270	30	5	6.00	0.17
5	120271	40	40	1.00	1.00
6	120272	20	20	1.00	1.00
7	120275	30	30	1.00	1.00
8	120660	50	50	1.00	1.00
9	120662	10	10	1.00	1.00
10	120663	0	5	0.00	

Modify the program to avoid division by zero notes.

ds04d05

61

Considerations

To avoid division by zero,

- if **FirstHalf** is 0, **Prop2=SecondHalf**
- if **SecondHalf** is 0, **Prop1=FirstHalf**.

62

Method 1: IF-THEN-ELSE Statements

DATA step IF-THEN-ELSE statements:

```
data work.ratio;
   format Prop1 Prop2 6.2;
   set orion.employee_donations(drop=Recipients Paid_By);
   FirstHalf=sum(Qtr1,Qtr2,0);
   SecondHalf=sum(Qtr3,Qtr4,0);
   if SecondHalf then Prop1=FirstHalf/SecondHalf;
      else Prop1=FirstHalf;
   if FirstHalf then Prop2=SecondHalf/FirstHalf;
      else Prop2=SecondHalf;
run;
```

ds04d05

63

Method 1: DS2 IF-THEN-ELSE Statements

DS2 DATA program IF-THEN-ELSE statements:

```
proc ds2;
data work.ratio_ds2/overwrite=yes;
   dcl double Prop1 Prop2 having format 6.1;
   dcl double FirstHalf SecondHalf;
   method run();
      set orion.employee_donations;
      FirstHalf=sum(Qtr1,Qtr2,0);
      SecondHalf=sum(Qtr3,Qtr4,0);
      if SecondHalf then Prop1=FirstHalf/SecondHalf;
         else Prop1=FirstHalf;
      if FirstHalf then Prop2=SecondHalf/FirstHalf;
         else Prop2=SecondHalf;
   end;
enddata;
run;
quit;
```

ds04d05

65

Method 1: DS2 IF-THEN-ELSE Statements

Partial DS2 DATA program IF-THEN-ELSE results:

Donation Ratio: IF-THEN-ELSE Statement
DS2 DATA Step

Obs	Employee_ID	FirstHalf	SecondHalf	Prop1	Prop2
1	120265	0	25	0.0	25.0
2	120267	30	30	1.0	1.0
3	120269	40	40	1.0	1.0
4	120270	30	5	6.0	0.2
5	120271	40	40	1.0	1.0
6	120272	20	20	1.0	1.0
7	120275	30	30	1.0	1.0
8	120660	50	50	1.0	1.0
9	120662	10	10	1.0	1.0
10	120663	0	5	0.0	5.0

ds04d05

66

Method 2: IFN Function

DATA step:

```
data work.ratio;
   format Prop1 Prop2 6.2;
   set orion.employee_donations(drop=Recipients Paid_By);
   FirstHalf=sum(Qtr1,Qtr2,0);
   SecondHalf=sum(Qtr3,Qtr4,0);
   Prop1=ifn(SecondHalf, FirstHalf/SecondHalf, FirstHalf);
   Prop2=ifn(FirstHalf, SecondHalf/FirstHalf, SecondHalf);
run;
```

IFN(*logical expression*, *value returned if true*
,*value returned if false*<,*value returned if missing*>)

IFN can reduce IF-THEN logic to a single statement.

ds04d06

67

Method 2: IFN Function

Partial DATA step results:

Donation Propensity
Base SAS DATA Step: IFN Function

Obs	Employee_ID	FirstHalf	SecondHalf	Prop1	Prop2
1	120265	0	25	0.00	25.00
2	120267	30	30	1.00	1.00
3	120269	40	40	1.00	1.00
4	120270	30	5	6.00	0.17
5	120271	40	40	1.00	1.00
6	120272	20	20	1.00	1.00
7	120275	30	30	1.00	1.00
8	120660	50	50	1.00	1.00
9	120662	10	10	1.00	1.00
10	120663	0	5	0.00	5.00

ds04d06

68

Method 2: IFN Function

Partial Log

```
NOTE: Division by zero detected at line 1414 column 20.
Prop1=0.00 Prop2=. Employee_ID=120265 Qtr1=. Qtr2=. Qtr3=. Qtr4=25
 FirstHalf=0 SecondHalf=25
_ERROR_=1 _N_=1

NOTE: Division by zero detected at line 1414 column 20.
Prop1=0.00 Prop2=. Employee_ID=120663 Qtr1=. Qtr2=. Qtr3=5 Qtr4=.
 FirstHalf=0 SecondHalf=5
_ERROR_=1 _N_=10
```

Data errors from division by zero are back!

ds04d05

69

Method 2: IFN Function – DS2 DATA program

Partial Log

```
ERROR: Float divide by zero
ERROR: General error
NOTE: PROC DS2 has set option NOEXEC and will continue
      to prepare statements.
```

In DS2, the data errors produce more severe consequences.

ds04d06

70

Idea Exchange

SAS Code

```
ratio=ifn(FirstHalf>0,SecondHalf/FirstHalf,.);
```

Partial SAS Log

```
NOTE: Division by zero detected at line 6 column 36.
Employee_ID=120265 Qtr1=. Qtr2=. Qtr3=. Qtr4=25 Recipients=Mitleid
International 90%, Save the Baby Animals 10% Paid_By=Cash or Check
FirstHalf=0 SecondHalf=25 Ratio=0.0% _ERROR_=1 _N_=1
```

How can you avoid the division by zero message?

71

Method 2: DS2 IF Expression

In DS2, the IF expression replaces the IFN function.

```
proc ds2;
data work.ratio_ds2_if/overwrite=yes;
   dcl double Prop1 Prop2 having format 6.2;
   dcl double FirstHalf SecondHalf;
   method run();
      set orion.employee_donations;
      FirstHalf=sum(Qtr1,Qtr2,0);
      SecondHalf=sum(Qtr3,Qtr4,0);
      Prop1=if SecondHalf then FirstHalf/SecondHalf
            else FirstHalf;
      Prop2=if FirstHalf then SecondHalf/FirstHalf
            else SecondHalf;
   end;
enddata;
run;
quit;
```

IF *expression-1* **THEN** *expression-2* **ELSE** *expression-3*

ds04d06

72

Method 2: DS2 IF Expression

Partial DS2 DATA program SAS log:

```
60  proc ds2;
61  data work.ratio_ds2_if/overwrite=yes;
62     dcl double Prop1 Prop2 having format 6.2;
63     dcl double FirstHalf SecondHalf;
64     method run();
65        set orion.employee_donations;
66        FirstHalf=sum(Qtr1,Qtr2,0);
67        SecondHalf=sum(Qtr3,Qtr4,0);
68        Prop1=if SecondHalf then FirstHalf/SecondHalf
69              else FirstHalf;
70        Prop2=if FirstHalf then SecondHalf/FirstHalf
71              else SecondHalf;
72     end;
73  enddata;
74  run;
NOTE: Execution succeeded. 124 rows affected.
```

No more division by zero messages!

ds04d06

73

Method 2: DS2 IF Expression

Partial DS2 DATA program results:

Donation Propensity
DS2 SAS DATA Step: IF expression

Obs	Employee_ID	FirstHalf	SecondHalf	Prop1	Prop2
1	120265	0	25	0.00	25.00
2	120267	30	30	1.00	1.00
3	120269	40	40	1.00	1.00
4	120270	30	5	6.00	0.17
5	120271	40	40	1.00	1.00
6	120272	20	20	1.00	1.00
7	120275	30	30	1.00	1.00
8	120660	50	50	1.00	1.00
9	120662	10	10	1.00	1.00
10	120663	0	5	0.00	5.00

ds04d06

74

Method 3: DS2 SELECT Expression

Partial DS2 DATA program:

```
method run();
   set orion.employee_donations;
   FirstHalf=sum(Qtr1,Qtr2,0);
   SecondHalf=sum(Qtr3,Qtr4,0);
   Prop1= select when (SecondHalf) FirstHalf/SecondHalf
          otherwise FirstHalf end;
   Prop2= select when (FirstHalf) SecondHalf/FirstHalf
          otherwise SecondHalf end;
end;
```

SELECT [(*expression*)]
WHEN (*expression*)|(*expression_result1*) *result_expression1*
[**WHEN** (*expression*)|(*expression_result2*) *result_expression2*]
[**OTHERWISE** *default_result_expression*]
END;

ds04d06

75

Method 3: DS2 SELECT Expression

Partial DS2 DATA program SAS log:

```
85  proc ds2;
86  data work.ratio_ds2_select/overwrite=yes;
87     dcl double Prop1 Prop2 having format 6.2;
88     dcl double FirstHalf SecondHalf;
89     method run();
90        set orion.employee_donations;
91        FirstHalf=sum(Qtr1,Qtr2,0);
92        SecondHalf=sum(Qtr3,Qtr4,0);
93        Prop1= select when (SecondHalf) FirstHalf/SecondHalf
94               otherwise FirstHalf end;
95        Prop2= select when (FirstHalf) SecondHalf/FirstHalf
96               otherwise SecondHalf end;
97     end;
98  enddata;
99  run;
NOTE: Execution succeeded. 124 rows affected.
```

Also avoids division by zero messages

ds04d06

76

Method 3: DS2 SELECT Expression

Partial DS2 DATA program results:

Donation Propensity
DS2 SAS DATA Step: SELECT expression

Obs	Employee_ID	FirstHalf	SecondHalf	Prop1	Prop2
1	120265	0	25	0.00	25.00
2	120267	30	30	1.00	1.00
3	120269	40	40	1.00	1.00
4	120270	30	5	6.00	0.17
5	120271	40	40	1.00	1.00
6	120272	20	20	1.00	1.00
7	120275	30	30	1.00	1.00
8	120660	50	50	1.00	1.00
9	120662	10	10	1.00	1.00
10	120663	0	5	0.00	5.00

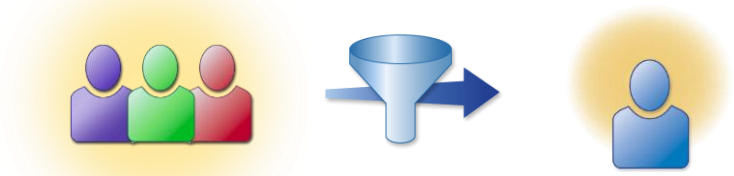
ds04d06

77

Business Scenario

The Orion Star charity campaign manager wants a report that contains a list of those employees who meet these two criteria:

- They have a propensity to donate in the first half of the year that is at least two times their propensity to donate in the second half of the year.
- They donate to an incorporated charity organization.



78

DS2 LIKE Expression

The *LIKE* expression compares character values to specified patterns. Two special 'wildcard' characters are used to define a text pattern.

- A percent sign (%) specifies that ***0 or more*** characters can occupy that position.
- An underscore (_) specifies that ***exactly one*** character must occupy that position.

Example	Selects
`where Name like '%n';`	All names ending in 'n'
`where Name like 'T_m';`	Tom and Tim, but not Thomas or Tammy
`where Name like 'T_m%';`	Tom, Tim, and Tammy, but not Thomas

79

DS2 LIKE Expression

Partial DS2 DATA program:

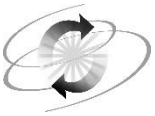
```
method run();
   set orion.employee_donations;
   FirstHalf=sum(Qtr1,Qtr2,0);
   SecondHalf=sum(Qtr3,Qtr4,0);
   Prop1=if SecondHalf then FirstHalf/SecondHalf
         else FirstHalf;
   Prop2=if FirstHalf then SecondHalf/FirstHalf
         else SecondHalf;
   if Prop1 >= 2*Prop2 and Recipients like '%Inc.%';
end;
```

Propensity 1 >= 2* Propensity 2
and donates to incorporated charity

Obs	Employee_ID	Prop1	Prop2	Recipients
1	120678	20.00	0.00	Disaster Assist, Inc. 50%, Cancer Cures, Inc. 50%
2	120813	2.00	0.50	Disaster Assist, Inc. 20%, Cancer Cures, Inc. 80%
3	121013	3.00	0.33	Child Survivors 60%, Disaster Assist, Inc. 40%
4	121018	10.00	0.00	Conserve Nature, Inc.
5	121093	5.00	0.00	Conserve Nature, Inc.
6	121125	10.00	0.00	Cancer Cures, Inc. 70%, Cuidadores Ltd. 30%
7	121128	10.00	0.00	Cancer Cures, Inc.
8	121131	10.00	0.00	Vox Victimas 40%, Conserve Nature, Inc. 60%
9	121133	5.00	0.00	Disaster Assist, Inc.

ds04d06

80



Exercises

Level 1

1. Determining Variable Types

a. Open program **ds04e01**. Complete the DS2 DATA program code to create the table **work.test**. The table should have only one observation and two variables. Declare these two global variables:

Number - type REAL
Text - type varchar(30)

b. Use the RUN method to do the following:

1) Assign the value *1234.5* to **Number**.

2) Assign the value *Testing* to **Text**.

c. Submit the PROC DS2 step and the PROC SQL step. Review the output to verify that there is only one row, and that the correct values are assigned to the variables.

Number	Text
1234.5	Testing

d. Submit the PROC CONTENTS step. Review the output. What is the actual variable type of the following?

Number _____

Text _____

Level 2

2. Adding Variables to a New Data Set

a. Open the **ds04e02** program. Add a DS2 DATA program to read in data from the **orion.Employees** table and create the **work.bonus** table.

b. Read only those rows from **orion.Employees** in which the uppercase value of **Country** is *AU* and the **Job_Title** column contains *Sales Rep*.

c. Add a new variable named **Bonus**.

d. Using a DS2 IF expression, assign a bonus of 500 dollars for residents of Melbourne, and 300 dollars for all others.

e. Submit the entire program, and review the output. There should be 70 rows in the **work.bonus** table.

Partial Output

Australian Sales Rep Bonuses

Row	Employee_Name	Country	City	bonus
1	Elvish, Irenie	AU	Sydney	$300
2	Ngan, Christina	AU	Melbourne	$500
3	Hotstone, Kimiko	AU	Sydney	$300

Challenge

3. Creating a New Data Set with Specific Variables in Uppercase

a. Open the **ds04e03** program.

b. Add a DS2 DATA program to create the **work.bonus** table by reading data from the **orion.Employee_Addresses** and **orion.Employee_Organization** tables.

c. Include only sales reps from Australia in the output table.

d. Using a DS2 IF expression, assign a bonus of 500 dollars for residents of Melbourne, and 300 dollars for all others.

The **work.bonus** table should contain only four variables (**Employee_Name**, **Country**, **City,** and **Bonus)** and 70 rows of data.

In the source tables, some **Country** values are not uppercase. In the output, all **Country** values must be uppercase.

Partial Output

Australian Sales Rep Bonuses

Row	Employee_Name	Country	City	bonus
1	Elvish, Irenie	AU	Sydney	$300
2	Ngan, Christina	AU	Melbourne	$500
3	Hotstone, Kimiko	AU	Sydney	$300

4.5 Selected Functions

Objectives

- Convert values between DS2 and Base SAS data types.
- Manipulate TIMESTAMP, DATE, and TIME values.
- Execute SQL statements from DS2 DATA programs.

84

Processing ANSI and SAS Values

- DATA step
  - Floating-point numeric only
  - SAS date, time, and datetime values
- DS2 can process both ANSI and SAS date, time, and datetime values.
  - contextual automatic conversion
  - functions for explicit conversion

85

Processing ANSI Date and Time Values

- Non-coercible values only automatically convert to text.
- DS2 date and time data types are non-coercible.
- DS2 provides functions for converting the following:
  - SAS DOUBLE date value to DS2 DATE
  - SAS DOUBLE time value to DS2 TIME
  - SAS DOUBLE datetime value to DS2 TIMESTAMP
  - DS2 DATE, TIME, or TIMESTAMP to corresponding SAS DOUBLE date, time, or datetime

DS2 DATE, TIME and TIMESTAMP cannot be directly used as DOUBLE values for processing as Base SAS date, time, or datetime values.

86

Selected Functions

The TO_DATE function

- accepts a SAS date numeric argument
- returns a DS2 date value.

Partial Program

```
dcl date DS2_Date;
method run();
   set orion.sas_datetimes;
   DS2_Date=to_date(SAS_Date);
   put SAS_Date= '|' SAS_Date yymmdd10.
      '|' DS2_Date=;
end;
```

TO_DATE(*SAS date expression*)

Partial SAS Log

```
SAS_Date=0 | 1960-01-01 |  DS2_Date=1960-01-01
SAS_Date=366 | 1961-01-01 |  DS2_Date=1961-01-01
SAS_Date=731 | 1962-01-01 |  DS2_Date=1962-01-01
```

ds04d09

87

Selected Functions

The TO_TIME function

- accepts a SAS time numeric argument
- returns a DS2 time value.

Partial Program

```
dcl time DS2_Time;
method run();
   set orion.sas_datetimes;
   DS2_Time=to_time(SAS_Time);
   put SAS_Time= '|' SAS_Time time8.
       '|' DS2_Time=;
end;
```

TO_TIME(*SAS time expression*)

Partial SAS Log

```
SAS_Time=0 | 0:00:00 | DS2_Time=00:00:00
SAS_Time=90 | 0:01:30 | DS2_Time=00:01:30
SAS_Time=43200 | 12:00:00 | DS2_Time=12:00:00
```

ds04d10

88

Selected Functions

The TO_TIMESTAMP function

- accepts a SAS datetime numeric argument
- returns a DS2 timestamp value.

Partial Program

```
dcl timestamp DS2_Timestamp;
method run();
   set orion.sas_datetimes;
   DS2_Timestamp=to_timestamp(SAS_datetime);
   put SAS_Datetime= '|' SAS_Datetime datetime19.
       '|' DS2_Timestamp=;
end;
```

TO_TIMESTAMP(*SAS datetime expression*)

Partial SAS Log

```
SAS_Datetime=0 |  01JAN1960:00:00:00| DS2_Timestamp=1960-01-01 00:00:00
SAS_Datetime=31622490 |  01JAN1961:00:01:30| DS2_Timestamp=1961-01-01 00:01:30
SAS_Datetime=63201600 |  01JAN1962:12:00:00| DS2_Timestamp=1962-01-01 12:00:00
```

ds04d11

89

Selected Functions

The TO_DOUBLE function

- accepts a DS2 date, time, or timestamp value
- returns a SAS numeric date, time, or datetime value.

Partial Program

```
data test/overwrite=yes;
   dcl double SAS_Date having format yymmdd10.;
   dcl double SAS_Time having format time.;
   dcl double SAS_Datetime having format datetime19.;
   keep SAS_Date SAS_Time SAS_Datetime;
   method run();
      set orion_db.datetimes;
      SAS_Date=to_double("Date");
      SAS_Time=to_double("Time");
      SAS_Datetime=to_double("Datetime");
   end;
enddata;
run;
```

TO_DOUBLE(*DS2 date, time, or datetime value*)

ds04d12

90

Selected Functions

TO_DOUBLE function results:

work.test

Obs	SAS_Date	SAS_Time	SAS_Datetime
1	1960-01-02	0:00:00	02JAN1960:00:00:00
2	1960-01-01	0:00:00	01JAN1960:00:00:00
3	1960-01-01	0:01:30	01JAN1960:00:01:30

ds04d12

91

Selected Functions

The INTDT function increments a DS2 date expression.

- The increment is specified in days.

```
proc ds2;
data _null_;
   method init();
      dcl date y z;
      y=date '2013-01-15';
      z=intdt(y,-21);
      put y= z=;
   end;
enddata;
run;
quit;
```

INTDT(*date expression, increment*)

SAS Log

```
y=2013-01-15 z=2012-12-25
```

ds04d13

92

Selected Functions

The INTTS function increments a DS2 timestamp expression.

- The increment is specified in seconds.

```
proc ds2;
data _null_;
   method init();
      dcl timestamp y z;
      y=timestamp '2011-03-01 16:51:30.00';
      z=intts(y,30);
      put y= z=;
   end;
enddata;
run;
quit;
```

INTTS(*timestamp expression, increment*)

SAS Log

```
y=2011-03-01 16:51:30 z=2011-03-01 16:52:00
```

ds04d14

93

Selected Functions

The SQLEXEC function

- executes an arbitrary FedSQL statement
- does not return a result set.

Partial Program

```
data _null_;
   method INIT();
      sqlexec('create table work.banks as
                  select * from orion.banks');
   end;
   enddata;
run;
```

SQLEXEC('*FedSQL statement*')

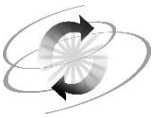
ds04d15

94

Executing FedSQL Statements with SQLEXEC

1. Open the **ds04d15** program.
2. Submit the code in Section A.
   a. Review the SAS log to verify that SQLEXEC created a new table, **work.banks**, with three rows.
   b. Review the procedure output to verify that **work.banks** contains three rows of data.
3. Submit the code in Section B and review the SAS log. This program attempts to create a table in the INIT method using SQLEXEC, then read the table with a SET statement in the RUN method. Because the table does not exist at compile time, the SET statement produces an error and the DS2 DATA program fails to compile. These are the resulting error messages:

```
ERROR: Compilation error.
ERROR: BASE driver, Table BANKS does not exist or cannot be accessed or created
ERROR: Table "WORK.BANKS" does not exist or cannot be accessed
ERROR: Line 53: Unable to prepare SELECT statement for table banks (rc=0x80fff802U).
```



Exercises

Level 1

4. Examining the Data Types

In 2011, necessary improvements to the shipping system for catalog orders were delayed to ensure that all items would be delivered in time for Christmas. Management wants a report that shows what the impact would be if shipping for these orders were delayed for five days.

a. Open the **ds04e04** program. Execute the code and review the log. Notice the error that indicates the invalid conversion of the date.

b. Scroll up in the log to review the results of the PROC SQL DESCRIBE TABLE query **orion_db.order_fact**. What data type is indicated for **Delivery_Date**? _____

c. Convert the PROC SQL statement to a PROC FEDSQL statement.

d. Execute the PROC FEDSQL query. What data type is indicated for **Delivery_Date**? _____

PROC SQL relies on the LIBNAME engine, so all numeric data types are interpreted as standard SAS numeric variables and all character types as standard fixed-width SAS character variables.
PROC FEDSQL can access the table metadata directly and reports the actual data types of the underlying variables.

e. In the DS2 DATA program, modify the line that increments the **Delivery_Date** value. Use the TO_DOUBLE function to convert **Delivery_Date** to a SAS date value before adding **5**.

```
New_Delivery_Date=to_double(Delivery_Date)+5;
```

f. Execute the modified DS2 DATA program. Review the SAS log to ensure that no errors were generated. The output should resemble the following:

Delivery Dates Delayed 5 Days

New_Delivery_Date	Customer_ID	Order_Date	Delivery_Date
03DEC2011	70187	23NOV2011	28NOV2011
20NOV2011	908	11NOV2011	15NOV2011

Level 2

5. Modifying a Program to Apply Specific Conditions

Management wants to evaluate the impact on customers if you switch to using guaranteed second-day delivery for all Internet orders.

Open the **ds04e05** program and modify the program so that the following conditions exist:

- The program produces no errors.
- New delivery dates are calculated for orders that were in transit for more than two days.
- Results are sorted by **Order_Date**.

The final report should resemble the following:

Web Orders Shipping Guaranteed 2nd Day Delivery

New_Delivery_Date	Customer_ID	Order_Date	Delivery_Date
.	46966	07APR2011	08APR2011
11APR2011	27	09APR2011	14APR2011
11APR2011	27	09APR2011	14APR2011
11APR2011	27	09APR2011	14APR2011
27APR2011	31	25APR2011	29APR2011

Challenge

6. Adding DATA program Code to Create Results Based on Conditions

a. Open the **ds04e06** program. Add DS2 DATA program code as indicated in the comments so that the following occurs:

1) If today is Monday, then all rows are deleted from **work.banks**.

2) Otherwise, write a note to the SAS log. Indicate the actual day of the week and that no rows were deleted from **work.banks**.

b. Submit the entire program. There should be no notes or errors in the log.

Depending on the day of the week, the report produced by the FedSQL step resembles one of these two possibilities:

- If today is Monday:

Rows in work.banks on Monday

column	COUNT
Before	3
After	0

- If today in ***not*** Monday:

Rows in work.banks on Wednesday

column	COUNT
Before	3
After	3

4.6 Solutions

Solutions to Exercises

1. Determining Variable Types

a. Open program **ds04e01**. Complete the DS2 DATA program code to create the table **work.test**. The table should have only one observation and two variables. Declare these two global variables:

Number - type REAL

Text - type varchar(30)

b. Use the RUN method to do the following:

1) Assign the value *1234.5* to **Number**.

2) Assign the value *Testing* to **Text**.

c. Submit the PROC DS2 step and the PROC SQL step. Review the output to verify that there is only one row, and that the correct values are assigned to the variables.

```
/*ds04s01*/
proc ds2;
data work.test;
   dcl real Number;
   dcl varchar(30) Text;
   method run();
      Number=1234.5;
      Text='Testing';
   end;
enddata;
run;
quit;

proc sql;
   select *
      from work.test
   ;
quit;

proc contents data=work.test;
run;
```

d. Submit the PROC CONTENTS step. Review the output. What is the actual variable type of the following?

Number	**Num (standard SAS floating-point numeric)**
Text	**Char (standard SAS fixed-width character variable)**

2. Adding Variables to a New Data Set

```
/*ds04s02*/

proc ds2;
data bonus /overwrite=yes;
```

```
   dcl double bonus;
   method run();
      set {select Employee_Name
                  ,Country
                  ,City
              from orion.Employees
              where country='AU'
                and Job_Title like '%Sales Rep%'};
      Bonus=if city='Melbourne' then 500
              else 300;
   end;
enddata;
run;
quit;

/*Use the NUMBER option to determine the number of rows returned*/
proc FedSQL number;
TITLE 'Australian Sales Rep Bonuses';
select Employee_Name
       ,Country
       ,City
       ,put(Bonus,dollar6.) as Bonus
   from bonus
;

/*Alternatively, use &SQLOBS to determine the number of rows
returned*/
%put ;
%put NOTE: &SQLOBS Rows were returned by the FedSQL query.;
%put;
quit;
title;
```

3. Creating a New Data Set with Specific Variables in Uppercase

```
/*ds04s03*/
proc ds2;
data bonus /overwrite=yes;
   dcl double bonus;
   method run();
      set {select Employee_Name
                  ,upcase(Country) as Country
                  ,City
              from orion.Employee_Addresses as a
                  ,orion.Employee_Organization as o
              where a.Employee_ID=o.Employee_ID
      /*PROC SQL CALCULATED keyword is invalid in FedSQL code*/
                and upcase(country)='AU'
                and Job_Title like '%Sales Rep%'};
      Bonus= Select(city)
                when ('Melbourne') 500
                otherwise 300
```

```
            end;
   end;
enddata;
run;
quit;

/*Use the NUMBER option to determine the number of rows returned*/
proc FedSQL number;
TITLE 'Australian Sales Rep Bonuses';
select Employee_Name
       ,Country
       ,City
       ,put(Bonus,dollar6.) as Bonus
   from bonus
;

/*Alternatively, use &SQLOBS to determine the number of rows
returned*/
%put ;
%put NOTE: &SQLOBS Rows were returned by the FedSQL query.;
%put;
quit;
title;
```

4. **Examining the Data Types**

```
/*ds04s04*/
proc fedsql;
   describe table orion_db.order_fact;
quit;

title "Delivery Dates Delayed 5 Days";
proc ds2;
data;
   dcl double New_Delivery_Date having format date9.;
   method run();
      set {select Customer_ID, Order_Date, Delivery_Date
              from orion_db.Order_Fact
              where delivery_date between date '2011-11-01'
                 and date '2011-11-30'
                 and order_type=2};
      New_Delivery_Date=to_double(Delivery_Date)+5;
   end;
enddata;
run;
quit;
title;
```

5. **Modifying a Program to Apply Specific Conditions**

```
 /*ds04s05*/
proc fedsql;
   describe table orion_db.order_fact;
```

```
quit;

title "Web Orders Shipping Guaranteed 2nd Day Delivery";
proc ds2;
data;
   dcl date New_Delivery_Date;
   method run();
      set {select Customer_ID, Order_Date, Delivery_Date
              from orion_db.Order_Fact
              where delivery_date between date '2011-04-01'
                 and date '2011-04-30'
                 and order_type=3
              order by Order_date, Delivery_Date};
      if to_double(Delivery_Date)-to_double(Order_Date) > 2
         then New_Delivery_Date=intdt(Order_Date,2);
   end;
enddata;
run;
quit;
title;
```

6. Adding DATA program Code to Create Results Based on Conditions

```
/*Create table work.banks*/
data work.banks /overwrite=yes;
   method run();
      set orion.banks;
   end;
enddata;
run;

data _null_;
/* Add code that deletes all rows from work.banks if today is Monday */
/* The required FedSQL statement is 'delete from work.banks'      */
   method init();
/* %tslit is required to single quote macro character constant values */
      /* The SQLEXEC function executes arbitrary FedSQL code      */
      if %tslit(&today)='Monday' then
         sqlexec('delete from work.banks');
      else
         put 'Today is' %tslit(&today) 'so no rows deleted.';
   end;
enddata;
run;
quit;

/* Report on number of rows before and after processing */
title "Rows in work.banks on &today";
proc fedsql;
   select 'Before', count(*) from orion.banks
      union all
   select 'After', count(*) from work.banks
```

```
      order by 1 desc
   ;
quit;
```

Solutions to Student Activities (Polls/Quizzes)

4.01 Poll – Correct Answer

Submit the program **ds04a01** and review the SAS log. Did the use of the DECIMAL data type produce the expected precision in the results?

- ○ Yes
- ◉ No

1. Constants are assumed to be DOUBLE.
2. Calculation produces a DOUBLE result.
3. DOUBLE is converted to DECIMAL on assignment to **c**.

```
***** DOUBLE TYPE *****
 123456789123456789 1234.56789 /  1234567891234599993344
 987654321987654321 9876.54321 /  9876543219876600217600
111111111111111111111111.11111 / 111111111111111000195072
```

22

4.02 Poll – Correct Answer

Modify all of the numeric constants in program **ds04a01** to use the **n** suffix. Resubmit the program and review the SAS log. Did the change to DECIMAL constants produce the expected precision in the results?

- ○ Yes
- ◉ No

1. Values are FORMATTED in the PUT statements.
2. SAS formats only apply to DOUBLE values.
3. DECIMAL is converted to DOUBLE for formatting.

```
***** DECIMAL TYPE *****
 123456789123456789 1234.56789 /  1234567891234568011776
 987654321987654321 9876.54321 /  9876543219876543594496
 111111111111111111111111.11111/ 111111111111111111344128
```

25

4.03 Multiple Choice Poll – Correct Answer

Submit the **ds04a02** program and review the SAS log. Why does the program not produce the correct answer?

a. The formula used to calculate the result is incorrect.
b. The results are formatted incorrectly.
c. The decimal variable **Ans** does not accommodate fractional values.
d. The numeric constant should use the **n** suffix.

DECIMAL(5) allows five digits of precision with zero digits to the right of the decimal point. To reserve three digits for fractional values, specify a scale of 3. For example:

```
dcl decimal (5,3) Ans;
```

30

4.04 Quiz – Correct Answer

Execute the **ds04a03** program and review the SAS log.

a. In the SAS data set **work.test**, what type is the variable **DesiredSalary**? **SAS 8-byte numeric**
b. In the Teradata table **my_db.test**, what type is the variable **DesiredSalary**? **Decimal**

When stored in a SAS data set, DS2 numeric data types map to SAS eight-byte floating-point numeric variables, and DS2 character data types map to fixed-width SAS character variables.

35

Chapter 5 Methods, Packages, and Threads

5.1 Methods

Objectives

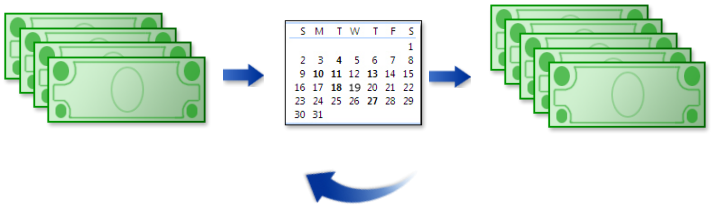
- Describe the purpose and uses of methods in DS2.
- Create a user-defined method.
- Create an overloaded user-defined method.

3

Business Scenario

Management wants a standardized method for estimating final values of accounts that accrue compound interest.

For annually compounded interest, the resulting value is equal to the sum of the starting value plus the starting value times the rate. The process should be repeated once for each year.



4

Business Scenario: Considerations

- The agreed-upon formula for the interest method is as follows:

  Amount=SUM(**Amount**, **Amount***Rate**)

- The code is reused in all DS2 DATA program programs that perform interest calculations.

5

Methods (Review)

Methods have the following characteristics:

- contain all executable code
- group related program statements into a single, reusable named unit
- leverage global and local variables

```
METHOD MethodName ([IN_OUT] parameter <, parameter,… > )
                                <RETURNS data-type >;
      … method-body …
END;
```

6

User-Defined Methods (Review)

User-defined methods have the following characteristics:

- can accept arguments
- can return a value
- execute when they are referenced (called) by name
- can be called multiple times

```
METHOD SumPlus (double V1,double V2 ) RETURNS double;
      RETURN (sum(v1,v2)+1);
END;
```

7

User-Defined Methods

Parameter Behavior

- The method can either modify one or more parameters at the call site or return a value.
- IN_OUT parameters are modified at the call site (similar to a DATA step CALL routine).

```
METHOD SumPlus (IN_OUT double V1,double V2 );
      V1=sum(v1,v2)+1;
END;
```

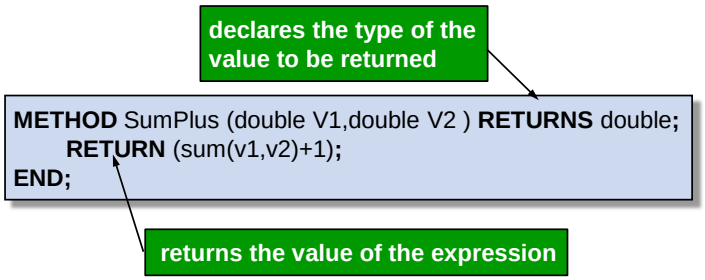
8

continued...

User-Defined Methods

Parameter Behavior

- If no IN_OUT parameters are defined, the method can return a value (like a DATA step function).



9

5.01 Multiple Choice Poll

How many parameters are required by a method calculating SUM(**Amount**,**Amount***Rate**) for a specified number of years?

a. 1
b. 2
c. 3
d. 4
e. other

10

Designing the Interest Method

- The interest method accepts three parameters.
  - amount
  - number of years
  - interest rate
- The value of **Amount** is modified at the call site.

  Amount=SUM(**Amount**, **Amount***Rate**)
  - The formula is executed once for every year.
- Other parameter values are not modified.

12

User-Defined Methods

This method accepts three arguments and modifies the value of the first parameter at the call site.

```
proc ds2;
data _null_;
   method interest(in_out double Amount,double Rate,int Years);
      dcl int i;
      do i=1 to Years;
         amount=sum(Amount,Amount*Rate);
      end;
   end;
   method init();
      dcl double Total;
      dcl double Duration;
      do Duration=1 to 5 by 2;
         Total=5000;
         interest(Total,0.04,Duration);
         put Duration= Total= dollar10.2;
      end;
   end;
enddata;
run;
quit;
```

```
Duration=1 Total= $5,200.00
Duration=3 Total= $5,624.32
Duration=5 Total= $6,083.26
```

ds05d02

13

Business Scenario

Management would like to modify the interest calculation method to add an additional parameter for the number of compounding periods during a year.

14

Business Scenario: Considerations

- Parameters are **Amount**, **Years**, **Rate**, and **Periods**.
- The agreed-upon formula for the new interest method is as follows:
  Amount=SUM(**Amount**, **Amount***(**Rate**/**Periods**))
- The formula should iterate once for every period in every year.

```
do i=1 to Years;
   do j=1 to Periods;
      Amount=sum(Amount,Amount*(Rate/Periods));
   end;
end;
```

15

Business Scenario: Considerations

- This method is to be used in all DS2 DATA programs that compound interest for multiple periods each year.
- The previously defined interest method is already adopted.
- User-defined methods cannot accept a variable number of arguments.

16

5.02 Poll

When you compound interest annually, three parameters are required. When you compound interest more frequently, four parameters are required.

Can you create a single interest method that can work in both of these scenarios?

- ○ Yes
- ○ No

17

Overloading Methods

- Methods can be overloaded.
- A method's *signature* is the ordered list of its parameter types.
- Overloading is accomplished by defining two or more same-named methods with different signatures.

```
method test(real v1) returns real;
   return (V1**2);
end;
method test(real v1, real v2) returns real;
   return (V1**V2);
end;
```

An error occurs if same-named methods are defined with identical signatures.

19

Overloading Methods

Calling an overloaded method

- Signatures are compared to parameters.
- The number of parameters must match a method signature.
  - The closest match by data types is selected.
  - Parameter data types auto-convert if necessary.
  - The method executes.
- An error occurs if any of the following conditions exist:
  - Method definition signatures are ambiguous.
  - The number of parameters, when called, does not match the number of parameters in any signature.
  - Data types do not match and are not coercible.

20

Overloaded Methods

1. Open the **ds05d03** program.
2. Review the DS2 Data program code. Notice the following:
   a. There are two methods named **interest** defined in the Data program.
   b. The first **interest** method has three parameters and calculates simple interest. Refer to this as the **simple interest** method. When it is executed, the method writes "Simple interest method" to the SAS log and modifies the **Amount** parameter at the call site.
   c. The second **interest** method has four parameters and calculates compound interest. Refer to this as the **compound interest** method. When it is executed, the method writes "Compound interest method" to the SAS log and modifies the **Amount** parameter at the call site.
   d. The INIT method has three DO loops.
      1) The first calls the **interest** method with three parameters and executes the **simple interest** method. After executing the method, this DO loop writes the number of years used for compounding and the resulting value to the SAS log.
      2) The second calls the **interest** method with four parameters and executes the **compound interest** method to calculate interest compounded quarterly. After executing the method, this DO loop writes the number of years used for compounding quarterly interest and the resulting value to the SAS log.
      3) The third calls the **interest** method with four parameters and executes the **compound interest** method to calculate interest compounded weekly. After executing the method, this DO loop writes the number of years used for compounding weekly interest and the resulting value to the SAS log.
3. Submit the program and review the SAS log. Notice the differences between the values that are calculated each time. From log entries, verify that the expected **interest** method actually executed.

Exercises

Level 1

1. Creating the Gross Margin Percentage Report

Management wants a report that indicates the gross margin percentage.

a. Open the **ds05e01** program. Before the RUN() method, create a user-defined method named **ourGMP** that calculates the gross margin percentage. The method must do the following:

1) Start with a METHOD statement defining method **ourGMP** as accepting two parameters (double **GS**, double **CoG**) and returning a double value.

2) In the body of the method, return the following value: (**GS**-**CoG**)/**GS**.

3) Close the method definition with an END statement.

b. Add code to the RUN method that executes the **ourGMP** method.

```
GrossMargin=ourGMP(GrossSales,CostOfGoods);
```

c. Run your program.

1) Review the SAS log to ensure that no warnings or errors were generated.

2) Review the output. The expected results are as follows:

Months with Gross Margin
Less than or equal to 49%

GrossMargin	YEAR	MONTH	GROSSSALES	COSTOFGOODS
47%	2007	4	2812	1486.85
47%	2010	9	1140.6	599.7
49%	2010	11	1402.1	716.9

Level 2

2. Creating a Report with Watch List Sales Items

Management wants a report that shows sales items that should be placed on a watch list. A sales item is on the watch list if the number of units sold is less than the target.

a. Open the **ds05e02** program and make changes.

b. Create a user-defined method named **ourTgt** that accepts two DOUBLE parameters and returns the target value. Calculate **Target** as follows:

Target=0.4/GrossMarginPercent

c. Use the **ourGMP** method from Exercise 1 to calculate the gross margin percentage. You can copy the code for the **ourGMP** method from the **ds05s01** program.

d. In the RUN method, the SET statement should use an SQL query to join **orion.order_fact** and **orion.product_dim** by **Product_ID**. Return the following computed values for the year 2010:

1) Columns (group by these):

   a) **Product_ID**, **Supplier_Name**, and **Product_Name**

2) Computed Columns

   a) **GrossSales**: sum of **Total_Retail_Price**

   b) **CostOfGoods**: sum of **CostPrice_Per_Unit*Quantity**

   c) **Units**: **SUM(Quantity)**

e. Run the entire program, including the PROC PRINT step. Verify the following:

1) No warnings or errors are generated in the SAS log.

2) The output produces the expected results.

Watch List Items

Supplier Name	Product Name	Product ID	Units Sold	Target	Gross Margin %	Gross Profit (Total)
Magnifico Sports	Rollerskate Roller Skates Gretzky Mvp S.B.S	240100400100	1	1.62737	25%	$38.00
Outback Outfitters Ltd	Comfort Shelter	230100700002	1	1.03529	39%	$85.00

Level 3

3. Creating a Marketing Model

The Marketing and Analysis statisticians created a marketing model. They used procedures in SAS High-Performance Analytics procedures to produce scoring code. They provided code for a DS2 method named SCORE. (See the **ds05e03** program. The beginning and end of the scoring algorithm code is clearly marked in the comments.)

Do not modify the scoring algorithm code.

a. Write a PROC DS2 Data program to score **orion.campaign** rows where **_PartInd_**=0. Use the user-defined method named **Score** to conduct the scoring. The method should accept one DOUBLE parameter (**ScoreVar**), which it modifies at the call site.

The final program must execute without producing errors or warnings in the SAS log when it is executed in strict variable declaration mode (SCOND=Error).

b. Complete the DS2 Data program. The score results should be stored in a DOUBLE variable named **FinalScore**. The results should be saved to **work.scored**, which contains only the variables **ID**, **target_b**, **_PartInd_**, and **FinalScore**.

c. The last section of **ds05e03** contains a PROC FEDSQL validation report, which ensures that the DS2 Data program scoring produced the correct results. Do not modify this code.

d. Run your finished program in its entirety.

1) Check the SAS log to ensure that no errors or warnings are generated.

2) Check the output and verify that the calculated validation number matches the expected value.

5.2 User-Defined Packages

Objectives

- Store user-defined methods in a DS2 package.
- Identify an appropriate package storage location.
- Execute user-defined methods from a package.

25

Business Scenario

Management wants the interest calculation methods that were previously agreed on to be used by all programmers in the company for every program that calculates interest.

26

Idea Exchange

How can the interest calculation method code be shared with all of the programmers in the company?

27

DS2 Packages

- DS2 packages are collections of methods and variables.
- DS2 packages can be used in DS2 programs and threads.
- Predefined packages are shipped with SAS and extend the capabilities of DS2.

28

DS2 User-Defined Packages

DS2 user-defined packages

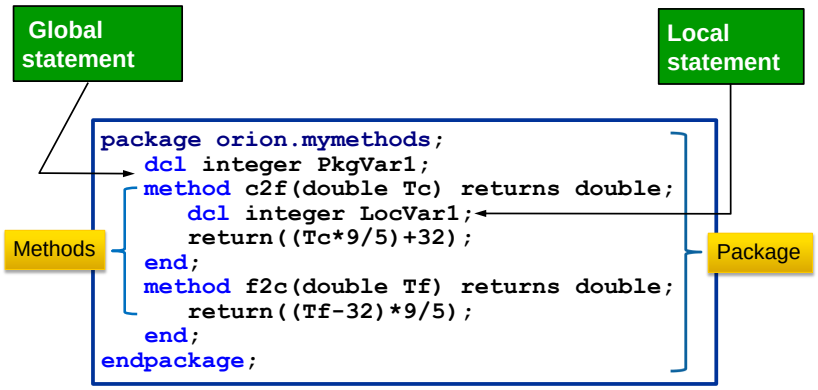
- enable users to extend DS2
- are collections of user-defined methods
- are stored in SAS libraries.
  - Work library for temporary (session) packages
  - permanent library for reusable packages

```
PACKAGE libref.package_name [/table-options];
    METHOD A();
        …method code…
    END;
    METHOD B();
        …method code…
    END;
ENDPACKAGE;
RUN;
```

29

DS2 PACKAGE Program Structure

- PACKAGE program block structure



30

User-Defined Packages

This package contains two METHOD definitions.

```
proc ds2;
package methods1 /overwrite=yes;
   method interest(in_out double Amount,double Rate,int Years);
      dcl int i;
      do i=1 to Years;
         Amount=SUM(Amount,Amount*rate);
      end;
   end;
   method interest(in_out double Amount,double Rate
                   ,int Years, int Periods);
      dcl int i j;
      do i=1 to Years;
         do j=1 to Periods;
            Amount=SUM(Amount,Amount*(Rate/Periods));
         end;
      end;
   end;
endpackage;
run;
quit;
```

```
NOTE: Created package methods1 in data set work.methods1.
```

ds05d04

31

5.03 Poll

The SAS log from the **ds05d04** program indicates that the package was created in the **work.methods1** data set. This package can be reused in subsequent SAS sessions.

- ○ True
- ○ False

32

Creating and Using a User-Defined Package

This demonstration illustrates using user-defined packages to permanently store, reuse, and share user-defined methods.

1. Examine the **ds05d05** program (Part A). Notice the following:

   a. The PACKAGE statement specifies that the package is stored in the permanent data set, **sasuser.methods1**.

   b. An overloaded method **contents()** which documents the methods available in this package.

      1) No parameters: writes a list of the methods to the SAS log.

      2) One parameter: Documents all of the package methods to the SAS log.

   c. The overloaded method **interest()**. There are three variants.

      1) No parameters: writes syntax to the SAS log; does not return a value.

      2) Three parameters: calculate annually compounded interest; return total investment value.

      3) Four parameters: calculate interest compounded multiple times per year; return total investment value.

   d. The ENDPACKAGE statement ends the package definition block of code. The RUN statement is required to execute this step.

   e. Submit Part A. Review the SAS log and verify that no errors or warnings are generated.

   f. Open the package data set, **sasuser.methods1**, and observe that the contents are merely encrypted source code.

	SAS_ROWID_	SAS_CHECKSUM_	SAS_TEXTPACKAGE_
1	1	18E9	DATA_STEP_2_PROGRAM
2	2	61E9	1.0
3	3	58E9	1312
4	4	41E9	JCP64AQGNE5TBAXFZIBHGKYU6THUYHURCCKIU3RSU4UE7MCI4SNV7C2GTJFIEVP7VYA5DLCGNJB3FRIHPPFXO
5	5	2E10	R3ELY4APN6XC75TJE2NFGGZYM57VD4INJ5DBTAQ3BICBRKH3CDN5EVYAZSJV6MIT6IDNURK5MDNH4Q5NRWDX
6	6	313E7	2RE6FT4R3I3CBAHOQDRHJRY6VGWIOXHAXPOUT4BB4RNJC5T4DG2NA5J7SB2BZW57PRCIBJ4IJVFEYFI3ZSKNQC
7	7	286E7	ZBBVVJDHFGXQ4S2VLEJWDLQ2ZUXUYZ6E46RE5QYWPLYY5QW46IKXZOSZRHCZBUS7CSDGYWKVDWPAWE4Y
8	8	24E9	QVTXA4GI73TSKOJXQWWRHS5SJRWYJCGOOTYMBDMHPEJH3ESFWZ5JV4PTMQ5OPMHBMZQ2GSKRDL46ZKDE
9	9	33E9	GNCPDVQEZEP5VH4VU7PMPMGVA6NWG6NLYT5RE7Y2EXIGOUBYGG4QFVFIZBCGVU6FEUOSPQMUTX6CWEWJ
10	10	16E9	6FTIKPRKUPMBQDXAO3KQVGBDLZ7CE42IC73XNZME3J6UD2ZMIVUKQ3AHL72FEPX2NGXHND33IDMIH6RVYWNT
11	11	57E9	KZ7MURW5GDZYOK6PGSU54FCUV4EK3Q5RSCQQMEF3GNFEXOKX64HQUTFBITTDNKX2DERS5WHI3CRYKSDVI
12	12	64E9	BR2FWWFP3TZ2SNEYP7BY3GV2YPAPDDYO2BP6EPEMBJ72FTTEH7GBCSXVV25KDJLJEVTTYKL44UFH2REYMN3
13	13	57E9	4DMSHGS3OFZX3SYW6OXKPERC3RORQHG7ILJM6HZDMWIEFPKN6IJV6IWUA7TLSI33N2A2VCIQLTPFGMBI6SW5
14	14	39E9	6U6QDCZCJYCGU3HQVMUHRXKJ6AQ6IWUROZ6BTNGQOXITZJVFEADPYC2HCOC3XBKSAJNXXG2UXTPTLLXTMD

   g. Close the viewer window.

2. Examine Part B of the **ds05d05** program. Notice the following:

   a. The global **declare** package instantiates the package **sasuser.methods1** as **m()**.

   b. The INIT method does the following:

1) executes the **contents()** method from package **m()** without parameters. This produces a list of available methods in the SAS log.

2) executes the **interest()** method from package **m()** without parameters. This produces syntax help for the **interest()** method in the SAS log.

c. Submit Part B. Review the SAS log and verify the following:

1) No errors or warnings are generated.

2) The expected method list and syntax help are generated.

3. Examine Part C of the **ds05d05** program. Notice the following:

a. The DS2 Data program reads from **orion.banks** and writes to **work.interest**.

1) The global **declare** package instantiates the package **sasuser.methods1** as **m()**.

2) The RUN method executes the **interest()** method from package **m()** in two DO loops.

a) The first calculates interest compounded annually for various investment durations, and outputs an observation for each calculation.

b) The second calculates interest compounded weekly for various investment durations, and outputs an observation for each calculation.

The PROC FedSQL step that lists the **work.interest** data set produced by the DS2 Data program.

b. Submit Part C.

1) Review the SAS log and verify that no errors or warnings are generated.

2) Review the output and verify that results appear as expected.

Projected values for $5,000 initial investment

Bank Name	Interest Rate	Years	Compounding	Total
Carolina Bank and Trust	0.0318	5	Annual	$5,847.20
National Savings and Trust	0.0328	5	Annual	$5,875.59
State Savings Bank	0.0321	5	Annual	$5,855.70
Carolina Bank and Trust	0.0318	5	Weekly	$5,861.40
National Savings and Trust	0.0328	5	Weekly	$5,890.77
State Savings Bank	0.0321	5	Weekly	$5,870.20

Exercises

Level 1

4. Modifying the Gross Margin Percentage Report

Modify the Gross Margin Percentage report process to use a package for the user-defined methods.

a. Open the **ds05e04** program.

b. In the first PROC DS2 step, create a package definition.

1) Add a PACKAGE statement to store the package as **work.pkg_05s04**.

2) Uncomment the method definition for **ourGMP**.

3) Add an ENDPACKAGE statement before the RUN statement.

c. In the second PROC DS2 step DATA statement, do the following:

1) Add a DECLARE package statement that instantiates **work.pkg_05s04** as **myPkg**.

```
dcl package work.pkg_05s04 myPkg();
```

2) Modify the statement that calculates **GrossMarginPercent** to use the **ourGMP** method from **myPkg**.

```
GrossMargin=myPkg.ourGMP(GrossSales,CostOfGoods);
```

d. Run the entire program.

1) Review the SAS log to ensure that no warnings or errors were generated.

2) Review the output. The expected results are as follows:

Months with Gross Margin Less than or equal to 49%

GrossMargin	YEAR	MONTH	GROSSSALES	COSTOFGOODS
47%	2007	4	2812	1486.85
47%	2010	9	1140.6	599.7
49%	2010	11	1402.1	716.9

Level 2

5. Modifying the Marginal Sales Items Reporting Process

Modify the marginal sales items reporting process to use a package.

a. Open the **ds05e05** program.

b. In the first PROC DS2 step, add a step that defines the package **work.pkg_05s05**, and contains the user-defined methods **Contents**, **ourGMP** and **ourBE**.

c. In the subsequent PROC DS2 Data programs, do the following:

1) Declare an instance of the package **work.pkg_05s05** as **myPkg**.

2) Test the **Contents** method.

3) Modify the **GrossMarginPercent** and **BreakEven** calculations to use the **ourGMP** and **ourBE** methods from **myPkg**.

d. Run your program.

1) Review the SAS log to ensure that no warnings or errors are generated, and that the **Contents** method listed the package methods.

2) Review the output. The expected results are as follows:

Marginal Sales Items

Supplier Name	Product Name	Product ID	UNITS	Break Even (Units)	Gross Margin %	Gross Profit (Total)
Mike Schaeffer Inc	Release Golf Sweatshirt w/Logo(1/100)	240200200080	2	2.56	90%	$168.80
Mike Schaeffer Inc	Top (1/100)	240200200081	2	2.56	90%	$193.10
Svensson Trading AB	Montana Adult Gloves	230100300013	1	1.44	80%	$19.80

5.3 Predefined Packages

Objectives

- Instantiate a predefined DS2 package in a DS2 DATA program.
- Execute methods from predefined DS2 packages.

38

Business Scenario

Orion Star developed many custom functions for DATA step programming using PROC FCMP. The FCMP function library is centrally maintained, and the functions are used by SAS programmers throughout the organization.

You want to use the functions in your DS2 programs, without incurring additional maintenance overhead.

39

FCMP Package

The FCMP package does the following:

- reads an FCMP function library
- produces a DS2 wrapper package

FCMP functions are executed as DS2 methods.

40

FCMP Package

Stipulations

- FCMP function arguments can be the following:
  - scalar DOUBLE, CHAR, NCHAR
  - arrays
- FCMP function OUTARGS parameters must be scalar.
- OUTARGS are IN_OUT parameters in DS2.
- DS2 passes numeric OUTARGS as DOUBLE.
- FCMP VARARGS functions are not supported.

41

Using the FCMP Package

This demonstration illustrates using the DS2 FCMP package to wrap FCMP functions for use in a DS2 Data program.

1. Open the **ds05d06** program. Execute the Setup portion (top of program to **Part A**).

   a. PROC SQL is used to create some macro variables that are used for future calculations.

   b. The PROC FCMP step creates two custom functions, **ourGMP** and **ourBE**.

2. Examine the Part A: DATA step / PROC SQL solution. Notice the following:

   a. The PROC SQL statement produces the intermediate data set **work.ProductInfo**, which contains sales information data from the years 2009 and 2010, summarized by product.

   b. The DATA step processes **work.ProductInfo** to calculate gross margin, break even, gross profit, and net revenue for each product. It keeps only the records where the units sold were less than the break-even number of units. The results are written to **work.underperforming_base**.

   c. The PROC PRINT step reads **work.underperforming_base** and produces a report about the underperforming products.

   d. Execute Part A and review the output.

Underperforming Sales Items - Base SAS Data Step

Supplier Name	Product Name	Product ID	Units Sold	Break Even (Units)	GrossSales	CostOfGoods	Fixed Costs	Gross Margin %	Net Revenue
Carolina Sports	Bat 5-Ply	240400200003	10000	10934	$62,000.00	$30,000.00	$34,987.20	52%	$-2,987.20
Nautlius SportsWear Inc	Capsy Hood	240100100031	5000	7289	$18,500.00	$6,500.00	$17,493.60	65%	$-5,493.60
Top Sports Inc	Baseball White Small	240700200007	5000	6033	$25,500.00	$11,000.00	$17,493.60	57%	$-2,993.60
Van Dammeren International	Tee Holder	240200100021	5000	11663	$13,500.00	$6,000.00	$17,493.60	56%	$-9,993.60

3. Examine the Part B: DS2 Data program solution. Notice the following:

   The first PROC DS2 step creates a DS2 package to wrap the FCMP function library that is located in **orion.fcmp_functions**. The DS2 package is stored as **orion.fcmp_funcs_ds2**.

   The subsequent DS2 Data program does the following:

   a. declares an instance of the **orion.fcmp_funcs_ds2** package named **myPkg**

   b. eliminates the PROC SQL step by incorporating the (slightly modified) SQL query in the SET statement

   c. calculates **BreakEven** and **GrossMargin** using the imported functions **ourGMP** and **ourBE** from package **myPkg**

   d. creates the **work.underperforming_ds2** data set, which contains only records where the units sold were less than the break-even number of units

   The PROC PRINT step produces a report about these underperforming products.

4. Execute Part B and review the output.

Underperforming Sales Items - DS2 Data Step

Supplier Name	Product Name	Product ID	Units Sold	Break Even (Units)	GROSSSALES	COSTOFGOODS	FIXED	Gross Margin %	Net Revenue
Carolina Sports	Bat 5-Ply	240400200003	10000	10934	62000	30000	34987.2	52%	$-2,987.20
Nautlius SportsWear Inc	Capsy Hood	240100100031	5000	7289	18500	6500	17493.6	65%	$-5,493.60
Top Sports Inc	Baseball White Small	240700200007	5000	6033	25500	11000	17493.6	57%	$-2,993.60
Van Dammeren International	Tee Holder	240200100021	5000	11663	13500	6000	17493.6	56%	$-9,993.60

5. What happens if you modify the FCMP functions? Examine Part C, Section 1: DS2 Data program. Notice the following:

   a. The PROC FCMP step modifies the custom function **ourBE** in the FCMP function library so that it no longer applies the CEIL function to the returned value. The results returned from this function now contain fractional (decimal) values.

   b. The subsequent Base SAS DATA and PROC PRINT steps reproduce the Underperforming Sales Items report.

   c. Execute Part C, Section 1. Review the output and notice that the **Break Even** column now contains fractional values instead of integers.

Underperforming Sales Items - Base SAS Data Step
ourBE returns fractions instead of CEIL value

Supplier Name	Product Name	Product ID	Units Sold	Break Even (Units)	GrossSales	CostOfGoods	Fixed Costs	Gross Margin %	Net Revenue
Carolina Sports	Bat 5-Ply	240400200003	10000	10933.50	$62,000.00	$30,000.00	$34,987.20	51.6%	$-2,987.20
Nautlius SportsWear Inc	Capsy Hood	240100100031	5000	7289.00	$18,500.00	$6,500.00	$17,493.60	64.9%	$-5,493.60
Top Sports Inc	Baseball White Small	240700200007	5000	6032.28	$25,500.00	$11,000.00	$17,493.60	56.9%	$-2,993.60
Van Dammeren International	Tee Holder	240200100021	5000	11662.40	$13,500.00	$6,000.00	$17,493.60	55.6%	$-9,993.60

6. Examine Part C, Section 2 - DS2 Data program. Notice the following:

   a. No new wrapper package is created.

   b. The subsequent DS2 Data program does the following:

      1) declares an instance of the **orion.fcmp_funcs_ds2** package named **myPkg**

      2) calculates **BreakEven** and **GrossMargin** using the imported functions **ourGMP** and **ourBE** from package **myPkg**

      3) creates the **work.underperforming_ds2** data set, which contains only records where the units sold were less than the break-even number of units.

   c. The PROC PRINT step produces a report about these underperforming products.

7. Execute Part C, Section 2. Review the output and notice that the **Break Even** column also contains fractional values instead of integers.

Underperforming Sales Items - DS2 Data Step
ourBE returns fractions instead of CEIL value

Supplier Name	Product Name	Product ID	Units Sold	Break Even (Units)	Gross Sales	Cost of Goods	Fixed Costs	Gross Margin %	Net Revenue
Carolina Sports	Bat 5-Ply	240400200003	10000	10933.50	$62,000.00	$30,000.00	$34,987.20	51.6%	$-2,987.20
Nautlius SportsWear Inc	Capsy Hood	240100100031	5000	7289.00	$18,500.00	$6,500.00	$17,493.60	64.9%	$-5,493.60
Top Sports Inc	Baseball White Small	240700200007	5000	6032.28	$25,500.00	$11,000.00	$17,493.60	56.9%	$-2,993.60
Van Dammeren International	Tee Holder	240200100021	5000	11662.40	$13,500.00	$6,000.00	$17,493.60	55.6%	$-9,993.60

Business Scenario

Orion Star programmers frequently need to update an RDBMS table with values from a SAS data set. They heard that the DS2 SQLSTMT package enables issuing dynamic SQL statements from a DS2 DATA program. They wonder if it could be used for this type of update.

43

SQLSTMT Package

This SQLSTMT package provides methods for the following:

- executing dynamically generated FedSQL statements
- interrogating the rows returned in the result set

Using an SQLSTMT package in your thread or data program precludes in-database processing.

44

Using the SQLSTMT Package

- Compile-time instantiation
  - declare an instance and define the SQL statement and parameter variables

```
dcl package sqlstmt update_me
   ('update orion_db.prices
       set Price=? where Product_ID=?'
       ,[B A]);
```

DECLARE PACKAGE SQLSTMT *variable*
('*sql-statement*' <,[*parameter-variable-list*]>);

If the tables used in the SQL statement do not exist at compile time, a syntax error results.

45

For example, the INIT() method step might use the SQLEXEC function to create the table **work.test**. In the same DS2 Data program, another method might contain an SQLSTMT package instance used to add data rows to the **work.test**. In this case, **work.test** does not exist at compile time.

Using the SQLSTMT Package

- Execution-time instantiation
  - declare an instance
  - instantiate later with the _NEW_ operator

```
dcl package sqlstmt update_me;
   update_me=_NEW_ SQLSTMT
     ('update orion_db.prices
        set Price=? where Product_ID=?'
        ,[B A]);
```

DECLARE PACKAGE SQLSTMT *variable*;
variable=_NEW_ SQLSTMT
('*sql-statement*'<,[*parameter-variable-list*]>);

46

Using the SQLSTMT Package

- FedSQL statement parameters assume the current value of the corresponding DS2 variable.

```
dcl package sqlstmt update_me
   ('update orion_db.prices
       set Price=? where Product_ID=?'
       ,[B A]);
```

47

Using the SQLSTMT Package

- A FedSQL statement can return a result set.

```
dcl package sqlstmt values
   ('select product_ID, New_Price
        from orion.price_updates');
```

Rows of the result set are accessed using the methods EXECUTE(), BINDRESULTS(), and FETCH().

48

Using the SQLSTMT Package

- Execute the SQL statement with the EXECUTE() method.

```
values.execute();
```

package_variable.execute();

- Bind results to DS2 variables with the BINDRESULTS() method.

```
values.bindresults([A,B]);
```

package_variable.bindresults([variable-list]);

49 ...

Using the SQLSTMT Package

- Fetch the next row of the result set with the FETCH() method.

```
rc=values.fetch();
```

package_variable.fetch();

Return Code	Meaning
0	Success
1	Error
2	No more data

50

Using the SQLSTMT Package

This demonstration illustrates using the DS2 SQLSTMT package to execute dynamic, data-driven SQL statements from a DS2 Data program, and update rows in an RDBMS table with values from a SAS data set.

1. Open the **ds05d07** program and execute Section A (Set up and pre-modification report). Review the report titled "Before Update." There should be 48 rows in this report.

   Partial Output

Before Update

Row	product_id	PRICE
1	240700100017	26.6
2	240500100062	55.1
3	240700400002	33.6
4	240300300071	138

2. Review the DS2 program in Section B (Update RDBMS table with SAS data set values). Notice the following:

   a. The first DECLARE PACKAGE statement instantiates the SQLSTMT package as **values** with a query that reads new product prices from the SAS data set, **orion.price_updates**. Values returned for **Product_ID** and **New_Price** are bound to the DS2 variables **A** and **B**, respectively.

   b. The second DECLARE PACKAGE statement instantiates the SQLSTMT package as **update_me**, with an SQL statement that updates table rows in the RDBMS table **orion_db.prices**. The statement also sets the values for **Price** to the value of the DS2 variable **B** where the value of **Product_ID** matches the value of the DS2 variable **A**.

   c. The call to the EXECUTE method of the **values** package executes the query.

   d. The call to the BINDRESULTS method of the **values** package binds the values returned to the DS2 variables **A** and **B**.

   e. The DO loop iteratively executes the **values** package FETCH method to access the result set rows one at a time. The FETCH method is normally called in a DO loop.

   f. The call to the EXECUTE method of the **update_me** package updates the appropriate row in the RDBMS table **orion_db.prices**.

3. Execute Section B and review the SAS log to ensure that no warnings or errors are generated.

4. Execute Section C (Post-modification report) and review the report titled "After Update." There should be 48 rows in this report.

   Partial Output

After Update

Row	product_id	PRICE
1	240700100017	27.67
2	240500100062	57.52
3	240700400002	35.05
4	240300300071	145.35

5. To restore original values to **orion_db.prices**, execute Section D (Reset original values and report) and review the report titled "After Reset." There should be 48 rows in this report, and the values should match those in the "Before Update" report.

   Partial Output

After Reset

Row	product_id	PRICE
1	240700100017	26.6
2	240500100062	55.1
3	240700400002	33.6
4	240300300071	138

Additional Predefined Packages

- Other DS2 predefined packages shipping with SAS include the following:
  - Hash/Hash Iterator
  - HTTP
  - JSON
  - Matrix
  - TZ (timezone)

52

The Hash Package

The HASH Package instantiates and manipulates hash objects. Hash lookups can be significantly faster than formats or array lookups. Speed varies with number of unique keys and table size. A hash object can do the following:

- Load data from a table
- Store and retrieve data in memory
- Replace, modify, and remove data in memory
- Output the hash data to a table

Using an initialized hash package in your thread or data program precludes in-database processing.

ds05d08

53

The HTTP Package (SAS9.4M2)

The HTTP package enable access to web services.

- Using the HTTP package you can
  - create and execute HTTP GET, HEAD, or POST methods
  - retrieve the response from the web server
  - set a socket time-out value
  - log the HTTP traffic between the HTTP client and server using the SAS logging facility.

Using an HTTP package in your thread or data program precludes in-database processing.

ds05d09

54

The JSON Package (SAS9.4M3)

The JSON package provides an interface to create and parse JSON text.

- Write methods accumulate write requests in memory
- In-memory text can be retrieved
- The parser enables you to read and parse JSON text

ds05d10

55

The Matrix Package

- Performs matrix operations in DS2
- Load matrices to and from DS2 arrays
- Add, subtract, multiply, divide
- Modular math and more

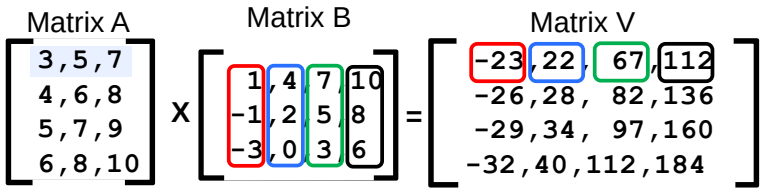
In a thread program, each thread has its own instance of the matrix and performs matrix operations on this instance.

56

The Matrix Package

Example: multiply matrix B by matrix A

- Multiply each element of each row of matrix A with the corresponding element in each column of matrix B.
- Summing the results of those multiplications produces the individual values stored in the elements of Matrix V




```
sum(3* 1,5*-1,7*-3)=-23
sum(3* 4,5* 2,7* 0)= 22
sum(3* 7,5* 5,7* 3)= 67
sum(3*10,5* 8,7* 6)=112
```

57 ds05d11

The TZ Package (SAS9.4M3)

The TZ package enables you to process local and international time and date values.

- The TZ package provides methods for
  - retrieving UTC time
  - retrieving local time
  - retrieving the time zone offset value.

58 ds05d12

For more information, see *Predefined DS2 Packages* in the DS2 documentation.

Exercises

Level 1

6. Updating a Column in a Report

Orion Star received the monthly updates to employee pay. The update data set, **orion.Pay_Updates**, contains three variables: **Employee_ID**, **Pay**, and **New_Pay**. Use the **New_Pay** values to update the **Salary** column of **orion_db.Employee_Payroll**.

a. Open the **ds05e06** program.

b. Find the DS2 Data program in Section C (Reset original values and report)

c. Copy the DS2 Data program code into Section B, immediately before the PROC SQL step.

d. Modify the DCL PACKAGE SQLSTMT statement for **values** to select **New_Pay** instead of **Pay** when you query **orion.Pay_Updates**.

e. Run Sections A and B of the program. Review the SAS log to ensure that no errors or warnings are generated.

f. Review the two generated reports to verify that the **Salary** values were properly modified, and only the salaries for employees *121086* and *121128* were affected.

Before Update

Row	Employee_ID	Salary
1	120101	$163,040.00
2	121086	$26,820.00
3	121128	$25,405.00
4	121148	$52,930.00

After Update

Row	Employee_ID	Salary
1	120101	$163,040.00
2	121086	$28,161.00
3	121128	$26,675.25
4	121148	$52,930.00

g. Run Section C of the program to restore the original **Salary** values. Verify that the output from the After Reset report matches the results of the Before Update report.

Before Update

Row	Employee_ID	Salary
1	120101	$163,040.00
2	121086	$26,820.00
3	121128	$25,405.00
4	121148	$52,930.00

After Reset

Row	Employee_ID	Salary
1	120101	$163,040.00
2	121086	$26,820.00
3	121128	$25,405.00
4	121148	$52,930.00

Level 2

7. Updating Values in a Data Set

Orion Star supplier Top Sports changed the names of some of its products. The new names are supplied in the **orion.Product_New_Names** data set. Use the SQLSTMT package to update the product names in **orion_db.Product_Dim**.

a. Open the **ds05e07** program. Execute Section A (Pre-modification report). Review the output and notice that the names of four of the five Top Trim products should be modified.

Top Trim Products - Names Before Update

Product_ID	Product_Name
240300300024	Men's Summer Shorts
240300300065	Top Men's Goretex Ski Pants
240300300070	Top Men's R&D Ultimate Jacket
240300300071	Top Men's Retro T-Neck
240300300090	Top R&D Long Jacket

Top Trim Products - Names to be Modified

Product_ID	Product_Name_New	Product_Name
240300300065	Men's Goretex Ski Pants	Top Men's Goretex Ski Pants
240300300070	Men's R&D Ultimate Jacket	Top Men's R&D Ultimate Jacket
240300300071	Men's Retro T-Neck	Top Men's Retro T-Neck
240300300090	R&D Long Jacket	Top R&D Long Jacket

b. Complete the DS2 DATA _NULL_ step to perform the update and use the SQLSTMT package. The required SQL statements are as follows:

1) query to retrieve new product names:

```
select Product_ID, Product_Name_New
   from orion.Product_Name_Updates
```

2) parameterized SQL statement to update **orion_db.product_dim**:

```
update orion_db.Product_dim
   set Product_Name=?
   where Product_ID=?
```

c. Submit your DS2 Data program program.

d. Review the SAS log to verify that no warnings or errors are generated

e. Submit the subsequent PROC SQL query to generate the report titled "Top Trim Products - Names After Update." Verify that the four products had their names properly modified, and that the name for product number *240300300024* remains unchanged.

Top Trim Products - Names After Update

Product_ID	Product_Name
240300300024	Men's Summer Shorts
240300300065	Men's Goretex Ski Pants
240300300070	Men's R&D Ultimate Jacket
240300300071	Men's Retro T-Neck
240300300090	R&D Long Jacket

f. Submit the %INCLUDE statement to reset the product names to their original values.

5.4 Threads

Objectives

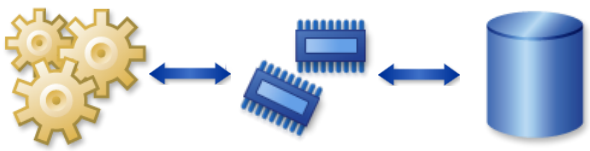
- Define threading for
  - input and output (I/O)
  - application processing.
- Rewrite a DS2 DATA program as a DS2 thread.
- Execute DS2 threads from a DS2 DATA program.

62

Processing Terminology

Task Processing

1. The program is loaded into memory.
2. Data is read from disk into memory.
3. The data in memory is processed.
4. The data in memory is written out to disk.



External I/O is ~1000x slower than in-memory processing.

63

Processing Terminology

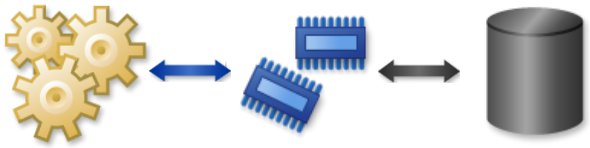
Single-Threaded Task Processing

- A single task (program) is processed start to finish.
- Multiple tasks are queued and processed sequentially.
- Disk I/O is approximately 1000 times slower than manipulating data in memory.
- The disparity between I/O and in-memory processing speeds produces bottlenecks.
- A process can be either CPU bound or I/O bound.

64

Processing Bottlenecks

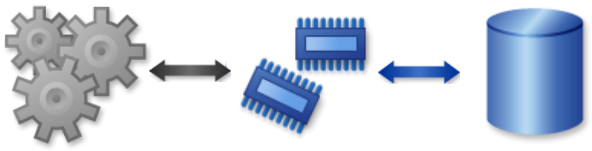
- I/O Bound
  - A process is I/O bound when computations are completed before additional data can be delivered.
  - I/O bound processes inefficiently use CPU.



65

Processing Bottlenecks

- CPU Bound
  - A process where data is delivered more quickly than it can be processed is CPU bound.
  - CPU bound processes are inefficient in I/O.



66

Processing Terminology

Threading

- Modern systems have multiple CPUs or CPUs that are capable of handling more than one process at a time in parallel.
- Each CPU process is called a *thread*.

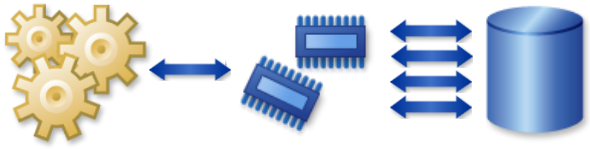
67

continued...

Processing Terminology

Threading

- Threaded I/O mitigates I/O bound processes.
  - This process delivers data records to the application in parallel.
  - CPU can process more continuously.



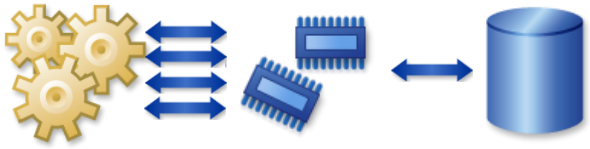
68

continued...

Processing Terminology

Threading

- Threaded application processing mitigates CPU-bound processes.
  - Computations are conducted on multiple data records in parallel.
  - I/O can proceed more smoothly.



69

Threading in Base SAS

Threaded I/O is commonplace.

- most modern operating system
- RDBMS
- massively parallel processing (MPP) databases

70

Threaded Applications

- Applications must be designed for multi-threading.
- Many SAS procedures are designed to use threads.
  - PROC SORT
  - PROC MEANS
  - PROC REPORT

  … and many others

71

Threading in Base SAS

DATA step

- The DATA step has threaded application processing for indexing (sort).
- Otherwise, processing is single threaded.

Single threaded, compute intensive applications can become CPU bound.

72

Threading in Base SAS

SAS/ACCESS Engines

- transparent Read and Write access to data sources
- threaded Read and Write access to and from most RDBMS
- implicit or explicit pass-through
  - The pass-through surfaces RDBMS data to SAS.
  - Data is then processed like native SAS data.
- The DATA step processes the data sequentially.
  - Processing might become CPU bound.

73

Threading in Base SAS: MPCONNECT

- An MPCONNECT session requires a SAS/CONNECT license.
- Use of MPCONNECT is a manual approach to DATA step threading.
- The programmer must write code to do the following:
  - split data
  - spawn SAS batch jobs to process each data set
  - monitor the process until all jobs are complete
  - process result data sets to produce the final result
- You can significantly accelerate CPU-bound processes.

74

Threading in Base SAS: DS2

- DS2 uses a driver versus a LIBNAME statement to connect to data.
- DS2 is designed for threaded application processing.
- DS2 threads
  - are easy to write
  - process multiple observations in parallel in a DATA program.

75

DS2 Threads

- Threads are similar to DS2 DATA programs.
  - can declare and use packages
  - can include INIT, TERM, and RUN methods
  - use a SET statement to accept input
  - are capable of BY-group processing, including First. and Last.
  - include global variables in output
- Threads cannot use the SET FROM statement.

76

DS2 Threads

Threads are similar to packages.

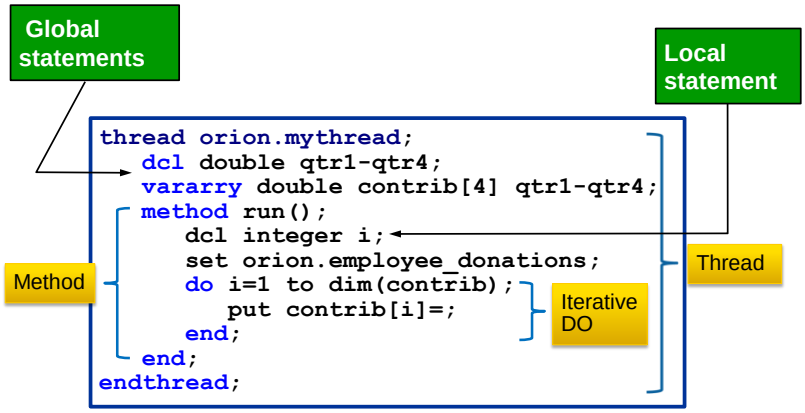
- can include user-defined methods
- can accept parameters
- are stored as tables in SAS libraries

Threads in permanent libraries can be reused.

77

DS2 Thread Program Structure

- THREAD program block structure



78

Business Scenario

An existing DATA step program is used to calculate quarterly employee charitable contributions, including Orion Star's 25% increase.

To run this process as multi-threaded in Base SAS, do the following:

- convert the DATA step to DS2
- convert the DS2 DATA program to a DS2 thread
- execute the thread from a DS2 DATA program

79

You can convert a DATA step directly to a DS2 thread program.

Convert a DATA Step to a DS2 DATA Program

Convert the global statements.

```
data new1;
   array contrib[*] qtr1-qtr4;


   set orion.employee_donations;
   where find(recipients,'%');


   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;

run;
```

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;









enddata;
run;
quit;
```

Global variables are written to the output data set by default.

80 ds05d14

Convert a DATA Step to a DS2 DATA Program

Convert the WHERE statement to SELECT.

```
data new1;
   array contrib[*] qtr1-qtr4;


   set orion.employee_donations;
   where find(recipients,'%');


   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;

run;
```

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();

   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
        from orion.employee_donations
        where find(recipients,'%')>0};




   END;
enddata;
run;
quit;
```

All executable code must be included in a method.

An embedded SELECT statement replaces the WHERE statement.

81 ds05d14

Convert a DATA Step to a DS2 DATA Program

Convert the remaining code.

```
data new1;
   array contrib[*] qtr1-qtr4;



   set orion.employee_donations;
   where find(recipients,'%');


   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;

run;
```

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
              , qtr2, qtr3, qtr4
        from orion.employee_donations
        where find(recipients,'%')>0};




   end;
enddata;
run;
quit;
```

q is a local variable and is not included in the output.

82 ds05d14

Convert a DATA Step to a DS2 DATA Program

Convert the remaining code.

```
data new1;
   array contrib[*] qtr1-qtr4;



   set orion.employee_donations;
   where find(recipients,'%');


   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   keep Employee_ID qtr: Total;

run;
```

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
              , qtr2, qtr3, qtr4
        from orion.employee_donations
        where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   END;
enddata;
run;
quit;
```

83 ds05d14

Convert a DATA Program to a Thread Program

Convert the DATA statement to a THREAD statement.

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
    from orion.employee_donations
    where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
enddata;
run;
quit;
```

```
proc ds2;
thread work.th_Donation/overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
       from orion.employee_donations
       where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
endthread;
run;
quit;
```

84 ds05d15

Convert a DATA Program to a Thread Program

Convert an ENDDATA statement to an ENDTHREAD statement.

```
proc ds2;
data new /overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
    from orion.employee_donations
    where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
enddata;
run;
quit;
```

```
proc ds2;
thread work.th_Donation/overwrite=yes;
   vararray double contrib[*] qtr:;
   dcl double Total;
   METHOD RUN();
   dcl int q;
   set {select Employee_ID, qtr1
             , qtr2, qtr3, qtr4
       from orion.employee_donations
       where find(recipients,'%')>0};
   Total=0;
   do q=1 to dim(contrib);
      Total+contrib[q]*1.25;
   end;
   end;
endthread;
run;
quit;
```

The THREAD program must be executed before the thread can be used in a subsequent DS2 DATA program.

85 ds05d15

A DATA step can be converted directly to a DS2 thread definition.

Using DS2 Threads

To use a thread in a DS2 DATA program, perform the following tasks:

- Declare an instance of the thread.
- Call the thread in a SET FROM statement.
  - Use the THREADS= option to specify the number of threads to execute.

Output rows are returned to the DS2 DATA program.

- Processing of thread results by additional statements before or after the SET FROM statement are single threaded.
- When you execute multiple threads, the order of rows that are returned might vary from run to run.

87

Using DS2 Threads

The THREADS= option specifies the number of threads to execute in parallel.

- Specifying more threads than available might negatively affect performance.

Without THREADS=, execution in Base SAS is single-threaded.

88

Using DS2 Threads

Execute the thread in a DATA statement

```
data new /overwrite=yes ;
   dcl thread work.th_Donation th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;
```

Declare a thread instance.

The SET FROM statement executes the thread.

Specify the number of threads.

92 ds05d15

Executing DS2 Threads in Base SAS

1. Open the **ds05d11a** program. The program contains the following:
   a. a DS2 Data program that is used to score the **orion.campaign** data set using a user-defined method named **Score**
   b. an SQL step that produces a verification report to ensure that scoring went as planned
2. Execute the program once. Verify the following:
   a. There are no warnings or errors in the SAS log.
   b. The verification report produces the expected results.
3. Modify the program to execute as a thread. The **ds05d11b** program contains the threaded version of the code for comparison.
   a. Change the DATA statement to a THREAD statement.

      Original:

```
data work.scored/overwrite=yes;
```

      Modified:

```
thread work.ScoreMe/overwrite=yes;
```

   b. Change the ENDDATA statement to an ENDTHREAD statement.
   c. After the PROC DS2 step, add an additional PROC DS2 step. The step should contain a DS2 Data program that declares an instance of the thread and executes it. (The code is available in the comments.)

```
proc ds2;
data work.scored_thread/overwrite=yes;
   dcl thread work.ScoreMe myThread;
   method run();
      set from MyThread;
   end;
enddata;
run;
quit;
```

4. Execute the program again. Verify the following:
   a. There are no warnings or errors in the SAS log.
   b. The results are still the same.
5. Add the option THREADS=4 to the DS2 Data program SET FROM statement.

   Original

```
set from MyThread;
```

Modified:

```
set from MyThread threads=4;
```

6. Execute the program again. Verify the following:
   a. There are no warnings or errors in the SAS log.
   b. The elapsed time for execution of the DS2 Data program is less than the CPU time.
   c. The results are still the same.
7. Modify the THREADS= option to specify a large number of threads.

   Original:

```
set from MyThread threads=4;
```

   Modified:

```
set from MyThread threads=124;
```

8. Execute the DS2 Data program again. Verify the following:
   a. There are no warnings or errors in the SAS log.
   b. The elapsed time for execution of the DS2 Data program is less than the CPU time.
9. Compare the elapsed time and CPU time to execution with THREADS=4. THREADS=4 should perform slightly better than THREADS=124, due to increased overhead that results from coordinating more threads than the system can process simultaneously.

Formal benchmarking in a realistic environment is recommended to determine the optimal THREADS= value for production DS2 processes.

Exercises

Level 1

8. Modifying a Report to Leverage DS2 Threads

Modify the Gross Margin Percent report process to leverage DS2 threads.

a. Open the **ds05e08** program.

b. Convert the DS2 Data program to a THREAD statement.

1) Change the DATA statement to a THREAD statement. The thread should be stored as **work.th_05s08**.

2) Change the ENDATA statement to the ENDTHREAD statement.

c. Add a second PROC DS2 step with a DATA statement.

1) Add a DECLARE THREAD statement that instantiates the thread **work.th_05s08**. Name this instance **th**.

2) In the RUN method, `set from th threads=4;` is the only required statement.

d. Run the entire program.

1) Review the SAS log to ensure that no warnings or errors are generated.

2) Review the output. The expected results are shown below.

Months with Gross Margin Less than or equal to 49%

GrossMargin	YEAR	MONTH	GROSSSALES	COSTOFGOODS
47%	2007	4	2812	1486.85
47%	2010	9	1140.6	599.7
49%	2010	11	1402.1	716.9

Level 2

9. Modifying a Report Process to Use DS2 Threads

Modify the marginal sales items reporting process to use DS2 threads.

a. Open the **ds05e09** program.

b. Change the PROC DS2 Data program to a THREAD definition. Save the thread as **work.th_05s09**.

c. Add a subsequent PROC DS2 Data program.

1) Declare an instance of the thread **work.th_05s09** as **th**.

2) In the RUN method, use a SET FROM statement to execute and use four threads.

d. Run your program.

1) Review the SAS log to ensure that no warnings or errors are generated.

2) Review the output. The expected results are as follows:

Marginal Sales Items

Supplier Name	Product Name	Product ID	UNITS	Break Even (Units)	Gross Margin %	Gross Profit (Total)
Mike Schaeffer Inc	Release Golf Sweatshirt w/Logo(1/100)	240200200080	2	2.56	90%	$168.80
Mike Schaeffer Inc	Top (1/100)	240200200081	2	2.56	90%	$193.10
Svensson Trading AB	Montana Adult Gloves	230100300013	1	1.44	80%	$19.80

5.5 Solutions

Solutions to Exercises

1. Creating the Gross Margin Percentage Report

```
 /*ds05s01*/
proc ds2;
title1 "Months with Gross Margin ";
title2 "Less than or equal to 49%";
data;
   dcl double GrossMargin having format percent6.2;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
   method run();
   set {select year(order_date) as Year
              ,month(order_date) as Month
              ,Sum(Total_Retail_Price) as GrossSales
              ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
         from orion.order_fact
         group by 1,2
         order by 1,2};
      GrossMargin=ourGMP(GrossSales,CostOfGoods);
      if GrossMargin <= .49;
   end;
enddata;
run;
title;
quit;
```

2. Creating a Report with Watch List Sales Items

```
 /*ds05s02*/
proc ds2;
data work.marginal /overwrite=yes;
   dcl double Product_ID having format 12. label 'Product ID';
   dcl double GrossMargin having format percent6.2
      label 'Gross Margin %';
   dcl double Score having format 5.2 ;
   dcl double GrossProfit having format dollar10.2
      label 'Gross Profit (Total)';
   dcl double Target;
   drop GrossSales CostOfGoods;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
   method ourTgt(double U, double GM) returns double;
      dcl double Tgt;
      Tgt=.4/gm;
      return(Tgt);
```

```
   end;
   method run();
      set {select Supplier_Name
                  ,o.Product_ID
                  ,Product_Name
                  ,Sum(Total_Retail_Price) as GrossSales
                  ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
                  ,Sum(Quantity) as Units
           from orion.order_fact as o
              , orion.product_dim as p
           where o.Product_ID=p.Product_ID
              and year(order_date)=2010
           group by 1,2,3
           order by 1,2};
      GrossMargin=ourGMP(GrossSales,CostOfGoods);
      Target=ourTgt(Units,GrossMargin);
      If Target >= Units;
      GrossProfit=GrossSales-CostOfGoods;
   end;
enddata;
run;
quit;

title "Watch List Items";
proc print data=work.marginal noobs label;
   var SUPPLIER_NAME PRODUCT_NAME Product_ID UNITS Target
       GrossMargin GrossProfit;
   label Units ='Units Sold';
run;
title;
```

3. Creating a Marketing Model

```
 /*ds05s03*/
proc ds2 scond=error;
data work.scored/overwrite=yes;
   keep TARGET_B ID _PartInd_ FinalScore;
   dcl double FinalScore;
   method Score(in_out double ScoreVar);
     /*ScoreVar is the target variable*/
      dcl char(12) I_TARGET_B  _ST12;
      dcl double
          p_target_b0 u_target_b _TEMP _P1 _P0 _IY _MAXP _LP0
          _DM_FIND _LMR_BAD _ST5 _2_5 _2_4 _2_3 _2_2 _2_1 _2_0
          _6_1 _6_0;
      _LMR_BAD = 0.0;
      if MISSING(IM_DEMMEDHOMEVALUE) then
         do;
            _LMR_BAD = 1.0;
            goto DefaultExit;
         end;
      if MISSING(IM_GIFTAVGCARD36) then
```

```
      do;
         _LMR_BAD = 1.0;
         goto DefaultExit;
      end;
   if MISSING(IM_GIFTCNT36) then
      do;
         _LMR_BAD = 1.0;
         goto DefaultExit;
      end;
   if MISSING(IM_GIFTTIMEFIRST) then
      do;
         _LMR_BAD = 1.0;
         goto DefaultExit;
      end;
   if MISSING(IM_GIFTTIMELAST) then
      do;
         _LMR_BAD = 1.0;
         goto DefaultExit;
      end;
   _2_0 = 0.0;
   _2_1 = 0.0;
   _2_2 = 0.0;
   _2_3 = 0.0;
   _2_4 = 0.0;
   _2_5 = 0.0;
   _ST5 = LEFT(TRIM(put(STATUSCAT96NK, $5.)));
   _DM_FIND = 0.0;
   if _ST5 <= 'F' then
      do;
         if _ST5 <= 'E' then
            do;
               if _ST5 = 'A' then
                  do;
                     _2_0 = 1.0;
                     _DM_FIND = 1.0;
                  end;
               else
                  do;
                     if _ST5 = 'E' then
                        do;
                           _2_1 = 1.0;
                           _DM_FIND = 1.0;
                        end;
                  end;
            end;
         else
            do;
               if _ST5 = 'F' then
                  do;
                     _2_2 = 1.0;
                     _DM_FIND = 1.0;
```

```
                    end;
                end;
            end;
         else
            do;
               if _ST5 <= 'N' then
                  do;
                     if _ST5 = 'L' then
                        do;
                           _2_3 = 1.0;
                           _DM_FIND = 1.0;
                        end;
                     else
                        do;
                           if _ST5 = 'N' then
                              do;
                                 _2_4 = 1.0;
                                 _DM_FIND = 1.0;
                              end;
                        end;
                  end;
               else
                  do;
                     if _ST5 = 'S' then
                        do;
                           _2_5 = 1.0;
                           _DM_FIND = 1.0;
                        end;
                  end;
            end;
         if ^_DM_FIND then
            do;
               _2_0 = .;
               _2_1 = .;
               _2_2 = .;
               _2_3 = .;
               _2_4 = .;
               _2_5 = .;
               _LMR_BAD = 1.0;
               goto DefaultExit;
            end;
         _6_0 = 0.0;
         _6_1 = 0.0;
         _ST12 = LEFT(TRIM(put(M_GIFTAVGCARD36, BEST12.)));
         if _ST12 = '0' then
            do;
               _6_0 = 1.0;
            end;
         else if _ST12 = '1' then
            do;
               _6_1 = 1.0;
```

```
      end;
   else
      do;
         _6_0 = .;
         _6_1 = .;
         _LMR_BAD = 1.0;
         goto DefaultExit;
      end;
   _LP0 = 0.0;
   _LP0 = _LP0 + (1.4083626419216E-6) * IM_DEMMEDHOMEVALUE;
   _LP0 = _LP0 + (-0.01129757026733) * IM_GIFTAVGCARD36;
   _LP0 = _LP0 + (0.07014728733418) * IM_GIFTCNT36;
   _LP0 = _LP0 + (0.00325352658053) * IM_GIFTTIMEFIRST;
   _LP0 = _LP0 + (-0.03942784854233) * IM_GIFTTIMELAST;
   _TEMP = 1.0;
   _LP0 = _LP0 + (0.28218821521876) * _TEMP * _6_0;
   _LP0 = _LP0 + (0.0) * _TEMP * _6_1;
   _TEMP = 1.0;
   _LP0 = _LP0 + (-0.04430991827625) * _TEMP * _2_0;
   _LP0 = _LP0 + (0.33212057095241) * _TEMP * _2_1;
   _LP0 = _LP0 + (-0.15280893542678) * _TEMP * _2_2;
   _LP0 = _LP0 + (-0.02184926539233) * _TEMP * _2_3;
   _LP0 = _LP0 + (0.08403485184956) * _TEMP * _2_4;
   _LP0 = _LP0 + (0.0) * _TEMP * _2_5;
   _TEMP = 0.0513406495515 + _LP0;
   if (_TEMP < 0.0) then
      do;
         _TEMP = EXP(_TEMP);
         _P0 = _TEMP / (1.0 + _TEMP);
      end;
   else _P0 = 1.0 / (1.0 + EXP(-_TEMP));
   _P1 = 1.0 - _P0;
   ScoreVar = _P0;
   _MAXP = _P0;
   _IY = 1.0;
   P_TARGET_B0 = _P1;
   if (_P1 > _MAXP + 1E-8) then
      do;
         _MAXP = _P1;
         _IY = 2.0;
      end;
   select (_IY);
      when (1.0)
         do;
            I_TARGET_B = '1';
            U_TARGET_B = 1.0;
         end;
      when (2.0)
         do;
            I_TARGET_B = '0';
            U_TARGET_B = 0.0;
```

```
            end;
         otherwise
            do;
               I_TARGET_B = '';
               U_TARGET_B = .;
            end;
      end;
      DefaultExit:
      if _LMR_BAD = 1.0 then
         do;
            I_TARGET_B = '';
            U_TARGET_B = .;
            ScoreVar = .;
            P_TARGET_B0 = .;
         end;
      ;
   end;
   method run();
      set  {select* from orion.Campaign where _partInd_=0 };
       /* Call the new user-defined method to score the data */
      Score(FinalScore);
   end;
enddata;
run;
quit;

/************************************************************
 Do not modify the validation report code below this line.
************************************************************/
title "Validation - Expected ASE is 0.241335";
proc FedSQL;
   select sum((FinalScore-Target_B)**2)/(count(*)) as ASE
      from work.Scored
   ;
quit;
title;
```

4. Modifying the Gross Margin Percentage Report

```
 /*ds05s04*/
proc ds2;
package work.methods_ds05s04 /overwrite=yes;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
endpackage;
run;
quit;

proc ds2;
title1 "Months with Gross Margin ";
title2 "Less than or equal to 49%";
```

```
data;
   dcl package work.methods_ds05s04 myPkg();
   dcl double GrossMargin having format percent6.2;
   method run();
   set {select year(order_date) as Year
              ,month(order_date) as Month
              ,Sum(Total_Retail_Price) as GrossSales
              ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
         from orion.order_fact
         group by 1,2
         order by 1,2};
      GrossMargin=myPkg.ourGMP(GrossSales,CostOfGoods);
      if GrossMargin <= .49;
   end;
enddata;
run;
title;
quit;
```

5. **Modifying the Marginal Sales Items Reporting Process**

```
 /*ds05s05*/
proc ds2;
package work.methods_ds05s05 /overwrite=yes;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
   method ourBE(double F, double GM) returns double;
      return F/GM;
   end;
endpackage;
run;
quit;

proc ds2;
data work.marginal /overwrite=yes;
   dcl package work.methods_ds05s05 myPkg();
   dcl double Product_ID having format 12. label 'Product ID';
   dcl double GrossMargin having format percent5.2
      label 'Gross Margin %';
   dcl double BreakEven having format 6.2
      label 'Break Even (Units)';
   dcl double GrossProfit having format dollar10.2
      label 'Gross Profit (Total)';
   drop GrossSales CostOfGoods;
   method run();
      set {select Supplier_Name
                 ,o.Product_ID
                 ,Product_Name
                 ,Sum(Total_Retail_Price) as GrossSales
                 ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
                 ,Sum(Quantity) as Units
```

```
          from orion.order_fact as o
             , orion.product_dim as p
          where o.Product_ID=p.Product_ID
             and year(order_date)=2010
          group by 1,2,3
          order by 1,2};
      /* Modified this code to use the ourGMP method from myPkg */
      GrossMargin=myPkg.ourGMP(GrossSales,CostOfGoods);
      /* Modified this code to use the ourBE method from myPkg */
      BreakEven=myPkg.ourBE(Units*1.15,GrossMargin);
      if BreakEven/Units le 1.5;
      GrossProfit=GrossSales-CostOfGoods;
   end;
enddata;
run;
quit;

title "Marginal Sales Items";
proc print data=work.marginal noobs label;
   var SUPPLIER_NAME PRODUCT_NAME Product_ID UNITS
       BreakEven GrossMargin GrossProfit;
   label Units ='Units Sold';
run;
title;
```

6. Updating a Column in a Report

```
 /*ds05s06*/
/* Section A: Set up and pre-modification report */
title "Before Update";
proc sql noprint;
select Employee_ID into :list separated by ','
   from orion.pay_updates;
reset print number;
select Employee_ID, Salary format=dollar12.2
   from orion_db.Employee_Payroll
   where Employee_ID in (120101,&list,121148)
   order by 1;
quit;
/*************************************************************
 Section B: Update RDBMS table with SAS data set values
            and report on results
 Use the SQLSTMT package to update orion_db.Employee_Payroll
 Salary values with the values from  orion.Pay_Updates New_Pay.
*************************************************************/
proc ds2;
data _null_;
   /* Declare global variable to hold bound values */
   declare decimal (13) EmpID;
   declare double Sal;
   method run();
      /* Instantiate sqlstmt package for source data query */
```

```
      /* Statement reads data from a SAS data set */
      dcl package sqlstmt values
         ('select Employee_ID, New_Pay from orion.pay_updates');
      /* Instantiate the sqlstmt package for update statement */
      dcl package sqlstmt update_me
      /* ? indicates value to be dynamically inserted */
      /* Values are updated in a Teradata table */
         ('update orion_db.Employee_Payroll set Salary=?
           where Employee_ID=?'
      /* [] positional variable list to supply dynamic values */
         ,[Sal EmpID]);
      /* Execute the values package FedSQL statement */
      values.execute();
      /* Positionally bind result set columns to variable list */
      values.bindResults([EmpID Sal]);
      /* Fetch a result set row - returns 0 if successful */
      do while (values.fetch() = 0);
         /* Execute the update statement */
         update_me.execute();
      end;
   end;
enddata;
run;
quit;

title "After Update";
proc sql number;
select Employee_ID, Salary format=dollar12.2
   from orion_db.Employee_Payroll
   where Employee_ID in (120101,&list,121148)
   order by 1;
quit;

/* Section C: Reset original values and report */
proc ds2;
data _null_;
   /* Declare global variable to hold bound values */
   declare decimal (13) EmpID;
   declare double Sal;
   method run();
      /* Instantiate SQLSTMT package for source data query */
      /* Statement reads data from a SAS data set */
      dcl package sqlstmt values
         ('select Employee_ID, Pay from orion.pay_updates');
      /* Instantiate the sqlstmt package for update statement */
      dcl package sqlstmt update_me
      /* ? indicates value to be dynamically inserted */
      /* Values are updated in a Teradata table */
         ('update orion_db.Employee_Payroll set Salary=?
           where Employee_ID=?'
         ,[Sal EmpID]);
```

```
      /* Execute the values package FedSQL statement */
      values.execute();
      /* Positionally bind result set columns to variable list */
      values.bindResults([EmpID Sal]);
      /* Fetch a result set row - returns 0 if successful */
      do while (values.fetch() = 0);
         /* Execute the update statement */
         update_me.execute();
      end;
   end;
enddata;
run;
quit;
title "After Reset";
proc sql number;
select Employee_ID, Salary format=dollar12.2
   from orion_db.Employee_Payroll
   where Employee_ID in (120101,&list,121148)
   order by 1;
quit;
title;
%symdel list;
```

7. **Updating Values in a Data Set**

```
 /*ds05s07*/
/* Section A: Pre-modification report */
proc sql;
title "Top Trim Products - Names Before Update";
select Product_ID, Product_Name
   from orion_db.Product_Dim
   where Product_Group ='Top Trim'
   order by 1;
title "Top Trim Products - Names to be Modified";
select *
   from orion.Product_Name_Updates
   order by 1;
quit;

/*************************************************************
Section B:
Use the SQLSTMT package to update orion_db.Product_Dim
Product_Name values with the values from Product_New_Name in
orion.Product_Name_Updates.

Query to retrieve new product names:
'select Product_ID, Product_Name_New from
orion.Product_Name_Updates'

Parameterized SQL statement to update orion_db.product_dim:
'update orion_db.Product_dim set Product_Name=? where Product_ID=?'
*************************************************************/
```

```
proc ds2;
data _null_;
   declare decimal (13) ProdID;
   declare char(45) New_Name;
   method run();
      dcl package sqlstmt values
         ('select Product_ID, Product_Name_New
              from orion.Product_Name_Updates');
      dcl package sqlstmt update_me
         ('update orion_db.Product_dim set Product_Name=?
              where Product_ID=?'
         ,[New_Name ProdID]);
      values.execute();
      values.bindResults([ProdID New_Name]);
      do while (values.fetch() = 0);
         update_me.execute();
      end;
   end;
enddata;
run;
quit;

proc sql;
title "Top Trim Products - Names After Update";
select Product_ID, Product_Name
   from orion_db.Product_Dim
   where Product_Group ='Top Trim'
   order by 1;
quit;

/* Run ds05s07r to reset original values when finished. */
%include "&path\ds05s07r.sas"/source2;
```

8. Modifying a Report to Leverage DS2 Threads

```
 /*ds05s08*/
proc ds2;
package work.methods_ds05s08 /overwrite=yes;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
endpackage;
run;

/* Convert this Data program to a thread definition */
thread work.threads_ds05s08/overwrite=yes;
   dcl package work.methods_ds05s08 myPkg();
   dcl double GrossMargin having format percent6.2;
   method run();
      set {select year(order_date) as Year
                  ,month(order_date) as Month
                  ,Sum(Total_Retail_Price) as GrossSales
```

```
                ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
             from orion.order_fact
             group by 1,2
             order by 1,2};
      GrossMargin=myPkg.ourGMP(GrossSales,CostOfGoods);
      if GrossMargin <= .49;
   end;
endthread;
run;
quit;

title1 "Months with Gross Margin ";
title2 "Less than or equal to 49%";
/* Add DS2 code here to execute the thread */
proc ds2;
data;
   dcl thread work.threads_ds05s08 rpt;
   method run();
      set from rpt threads=4;
   end;
enddata;
run;
quit;
title;
```

9. Modifying a Report Process to Use DS2 Threads

```
 /*ds05s09*/
proc ds2;
package work.methods_ds05s09 /overwrite=yes;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
   method ourBE(double F, double GM) returns double;
      return F/GM;
   end;
endpackage;
run;
thread work.thread_ds05s09/overwrite=yes;
   dcl package work.methods_ds05s05 myPkg();
   dcl double Product_ID GrossMargin BreakEven GrossProfit;
   drop GrossSales CostOfGoods;
   method run();
      set {select Supplier_Name
                ,o.Product_ID
                ,Product_Name
                ,Sum(Total_Retail_Price) as GrossSales
                ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
                ,Sum(Quantity) as Units
          from orion.order_fact as o
             , orion.product_dim as p
          where o.Product_ID=p.Product_ID
```

```
            and year(order_date)=2010
          group by 1,2,3
          order by 1,2};
      GrossMargin=myPkg.ourGMP(GrossSales,CostOfGoods);
      BreakEven=myPkg.ourBE(Units*1.15,GrossMargin);
      if BreakEven/Units le 1.5;
      GrossProfit=GrossSales-CostOfGoods;
   end;
endthread;
run;
quit;

/* Add DS2 code here to execute the thread */
proc ds2;
data work.marginal/overwrite=yes;
   dcl thread work.thread_ds05s09 th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;

title "Marginal Sales Items";
proc print data=work.marginal noobs label;
   var SUPPLIER_NAME PRODUCT_NAME Product_ID UNITS BreakEven
       GrossMargin GrossProfit ;
   label Units ='Units Sold'
         Product_ID ='Product ID'
         GrossMargin ='Gross Margin %'
         BreakEven ='Break Even (Units)'
         GrossProfit ='Gross Profit (Total)';
   format Product_ID 12. GrossMargin percent5.2 BreakEven 6.2
          GrossProfit dollar10.2;
run;
title;
```

Solutions to Student Activities (Polls/Quizzes)

5.01 Multiple Choice Poll – Correct Answer

How many parameters are required by a method calculating SUM(**Amount**,**Amount***Rate**) for a specified number of years?

a. 1
b. 2
c. 3
d. 4
e. other

One parameter is required for each variable in the equation, and one to specify the number of years.

11

5.02 Poll – Correct Answer

When you compound interest annually, three parameters are required. When you compound interest more frequently, four parameters are required.

Can you create a single interest method that can work in both of these scenarios?

○ Yes
◉ No

DS2 methods do not accept a variable number of parameters.

18

5.03 Poll – Correct Answer

The SAS log from the **ds05d04** program indicates that the package was created in the **work.methods1** data set. This package can be reused in subsequent SAS sessions.

- ○ True
- ◉ False

Because the package was saved to the Work library, the package is deleted when the SAS session terminates.

33

Chapter 6 DS2 Unleashed

6.1 SAS In-Database Code Accelerator

Objectives

- Leverage the SAS In-Database Code Accelerator to execute DS2 code outside of the SAS session.

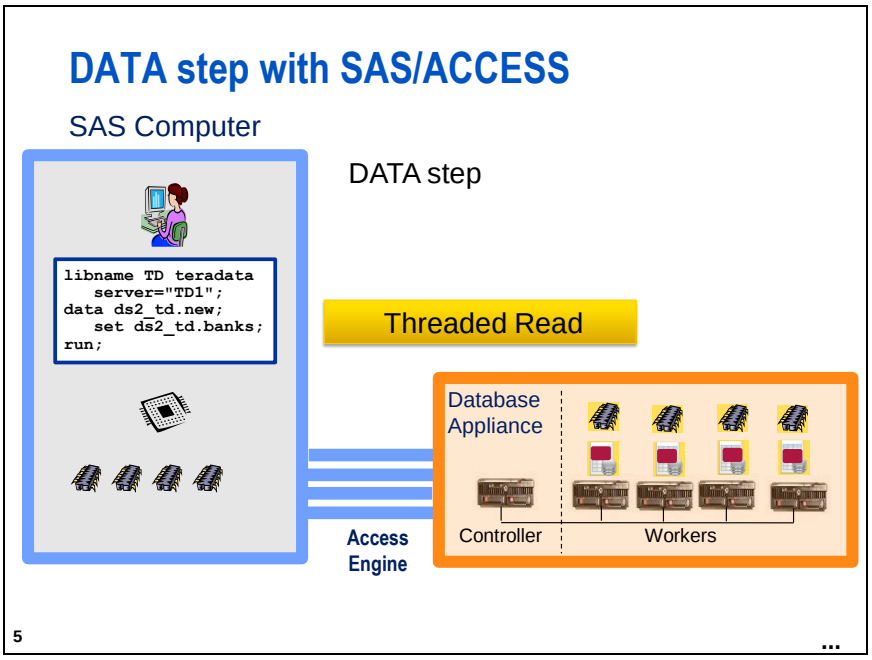
3

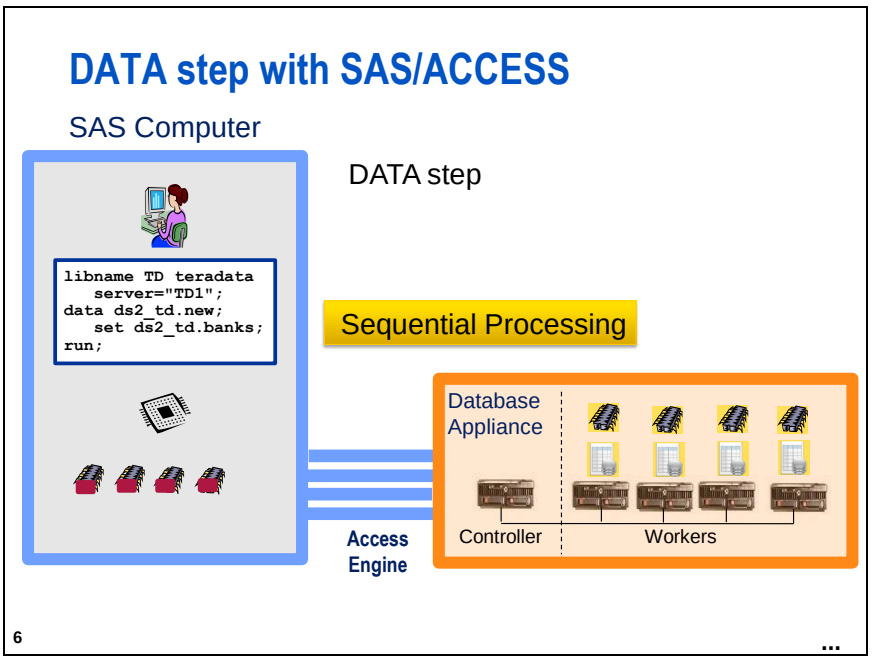
SAS In-Database Code Accelerator

Review of DATA step processing

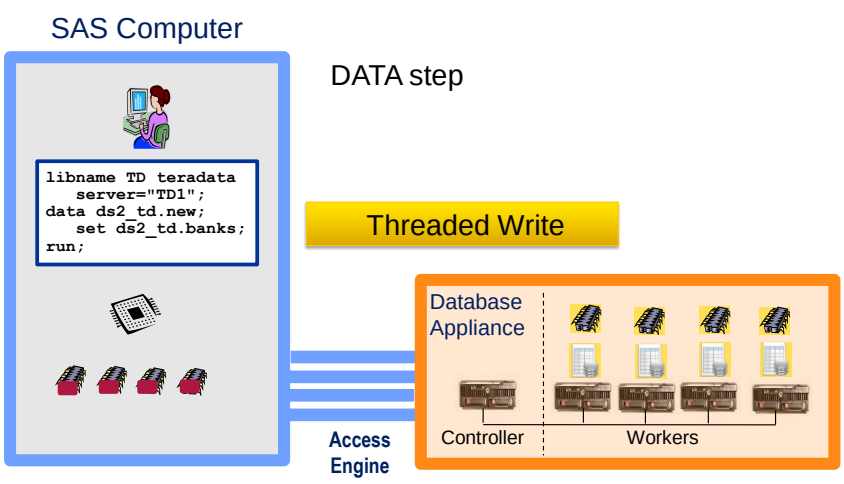
- Threaded reads occur with SAS/ACCESS to databases using implicit pass-through.
- Data moves from RDBMS to SAS server memory. (Multiple connections are possible to speed access.)
- Sequential processing of data in SAS occurs.
- Computationally intensive processing might become CPU bound.
- Data moves from SAS server memory to RDBMS. (Multiple connections are possible to speed access.)

4 ...





DATA step with SAS/ACCESS



7 ...

Business Scenario

Orion Star runs several computationally intensive production SAS programs against very large RDBMS tables. Management is considering licensing SAS In-Database Code Accelerator for your RDBMS.

You were tasked with investigating and reporting on the following:

- possible benefits of this product
- level of effort required for programmers to make DS2 programs execute in-database

8

SAS In-Database Code Accelerator

- The SAS In-Database Code Accelerator product is added to the database SAS/ACCESS Interface.
- The embedded process (EP) components are installed on the database appliance.
- DS2 threads are sent in-database as code, where they compile and execute in-database.
- For Hadoop and Teradata, the DS2 DATA program can also execute in-database if the output is a database table.
- Execution leverages RDBMS parallel processing.

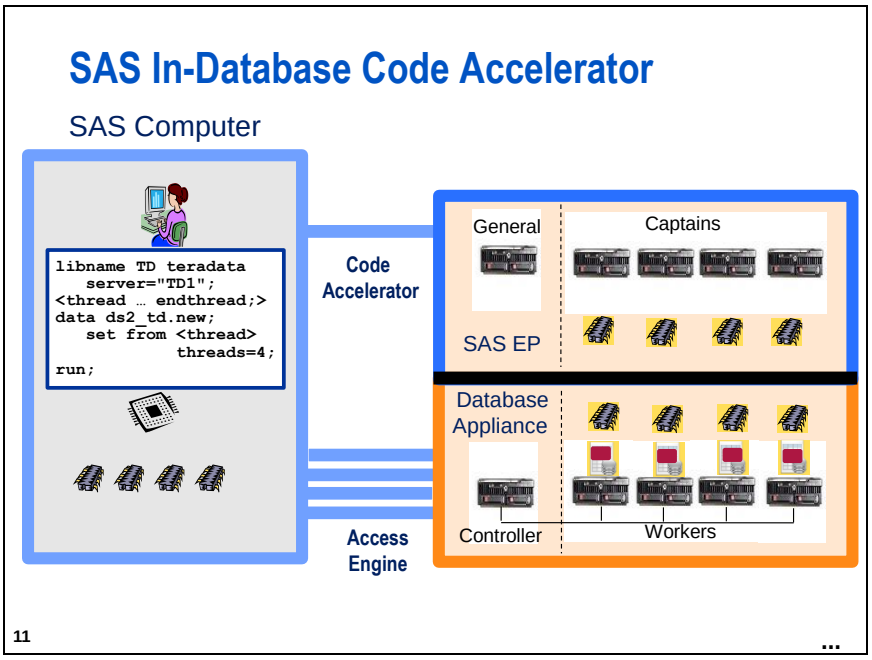
9 ...

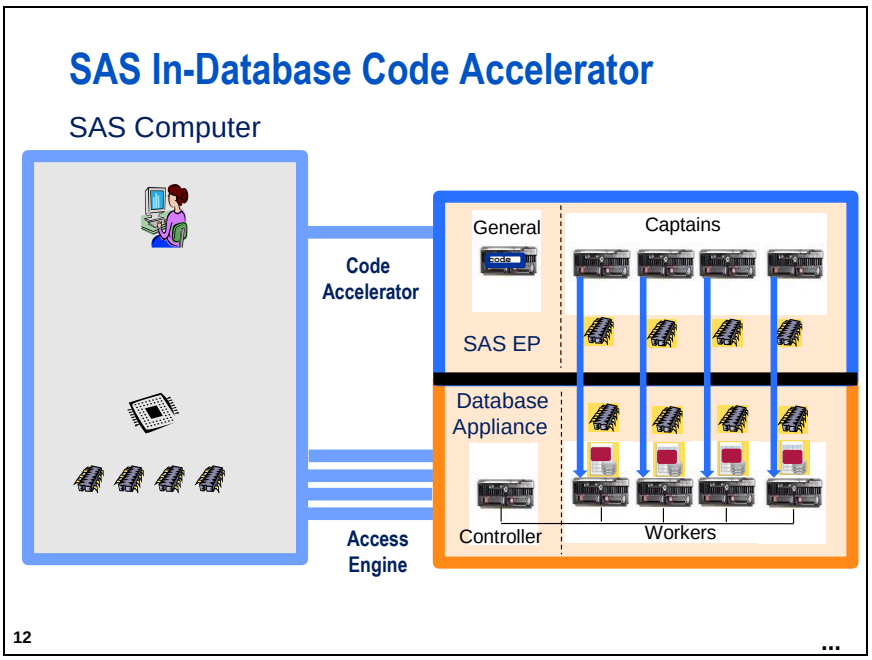
SAS In-Database Code Accelerator Limitations

- When the following methods are used to load data in a thread program, both the DATA program and thread programs are executed on the client, not in-database:
  - multiple SET statements
  - an SQLSTMT package
  - a Hash package
  - an HTTP package
  - In versions prior to SAS9.4M3
    - a SET statement with embedded SQL code
    - a SET statement reading multiple data sets

Matrix packages in thread programs produce a separate matrix instance for each thread.

10 ...





DS2 Unleashed

DS2 in-database processing does the following:

- decreases data movement
- computes in parallel on nodes containing data
- threads both I/O and application processing

DS2 threads can process in-database. For Teradata and Hadoop, the DS2 DATA program can also execute in-database.

13 ...

For a thread program to execute in-database, the SET statement in the thread must read from a table in a database with SAS Embedded process installed, and in-database execution must be enabled using the DS2ACCEL option.

For a DATA program to run in-database, it must SET FROM a thread running in-database, and the output dataset must use the same LIBREF as the table from which the thread is reading.

DS2 Unleashed

To execute in-database, use the ds2accel=yes option…

```
proc ds2 ds2accel=yes;
   thread work.th_Donation/overwrite=yes;
   dcl double Total;
   method run();
/* Runs threaded in Base SAS if libref for */
/* a database without SAS In-Database Code Accelerator */
      set orion.employee_donations;
      Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
endthread;
run;
data work.new (overwrite=yes) ;
   dcl thread work.th_Donation th;
   method run();
      set from th threads=4;
   end;
enddata;
run;
quit;
```

14 ds06d01 ...

There is an associated DS2ACCEL system option. Default value is DS2ACCEL=NO. For backward compatibility with older versions of PROC DS2, INDB is an alias for DS2ACCEL.

DS2 Unleashed

… and change the libref in the THREAD definition.

```
proc ds2 ds2accel=yes;
   thread work.th_Donation/overwrite=yes;
   dcl double Total;
   method run();
/* Runs in-database if libref for a database */
/* with SAS In-Database Code Accelerator installed  */
      set orion_db.employee_donations;
      Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
endthread;
run;
data work.new (overwrite=yes) ;
   dcl thread work.th_Donation th;
   method run();
/* THREADS= is ignored for in-database processing */
      set from th;
   end;
enddata;
run;
quit;
```

Threads= no longer required

15 ds06d02

Running DS2 code in-database

1. Open program **ds06d02** and execute the section labeled "Part A - Create package and thread". Verify the package **work.scoring** and the thread **my_db.th_Score** are created and that there are no errors or warnings in the SAS log.
2. Review the section of code labeled "Part B - Threaded alongside". Note that this DS2 DATA program and thread will process on the SAS compute platform because the PROC DS2 option DS2ACCEL=NO is set. Execute this section and verify there are no errors or warnings in the log. Make note of the elapsed time for processing.
3. Review the section of code labeled "Part C - Threaded in-database". Note that this DS2 DATA program and thread will execute in-database in Teradata the PROC DS2 option DS2ACCEL=YES is set. Execute this section and verify there are no errors or warnings in the log. Note that the SAS log indicates both the thread and DATA program executed in-database. Compare the elapsed time for processing in-database to the elapsed time for processing alongside.
4. Review the section of code labeled "Part D - Tracing DS2 in-database processing". Note the DS2_OPTIONS TRACE statement. Note that this statement only has an effect when processing in-database, and that it will produce a trace of all statements executed in the database by DS2. This option only remains in effect for the DATA program immediately following. Execute this section and verify there are no errors or warnings in the log. Note the extra information produced in the SAS log.
5. Review section of code labeled "Part E - Cleanup". Note the DS2 global statements DROP PACKAGE and DROP THREAD, which are not valid in the context of a DS2 program. Execute the code to delete the thread, package and results data sets.

6.01 Multiple Choice Poll

Which of these is ***not*** required to execute a DS2 thread program in an RDBMS?

a. Base SAS
b. SAS In-Database Code Accelerator for the RDBMS
c. SAS Embedded Process components installed on an RDBMS system
d. SAS High-Performance Analytics Server
e. SAS/ACCESS for the RDBMS

18

Exercises

Level 1

1. Executing DS2 Code in-database

a. Open the **ds06e01** program. Execute the program and review the SAS log.

Why didn't the THREAD program execute in-database?

_____

b. Add the DS2ACCEL=YES option to the PROC DS2 statement, and execute the program once again.

1) In the PROC statement, do the following:

a) Did the THREAD program execute in-database? _____

b) What changes would be required to make the DATA program run in-database?

_____

Level 2

2. Executing DS2 Code in-database

a. Open the **ds06e02** program.

b. After the PACKAGE program, add a THREAD program designed to score the data in my_db.campaign.

c. After the THREAD program, add a DATA program to execute the thread and write the results to a dataset named ds06s02_scored. Store the data in an appropriate library.

d. Execute the program, and review the SAS log to ensure:

1) No warnings or errors were generated.

2) Both the THREAD and DATA programs executed in-database.

6.2 Introduction to PROC HPDS2

Objectives

- Execute DS2 code in the SAS High-Performance Analytics grid using PROC HPDS2.
- Compare the capabilities of PROC HPDS2 versus PROC DS2.

PROC HPDS2 can run in single-machine mode. A SAS High-Performance Analytics grid is not available during class.

21

Business Scenario

Orion Star's SAS High-Performance Analytics grid is capable of executing HPDS2 code. Management requests that you investigate and compare the following:

- PROC HPDS2 and the SAS High-Performance Analytics grid versus PROC DS2 and the SAS in-Database Code Accelerator
- capabilities of PROC DS2 versus PROC HPDS2
- the level of effort for converting PROC DS2 DATA programs to execute in the SAS High-Performance Analytics grid using PROC HPDS2

22

Introduction to PROC HPDS2

- PROC HPDS2 executes DS2 code on massively parallel processing (MPP) platforms.
- The procedure is similar to PROC DS2.
  - can run threaded in SAS or in-database
  - DS2 DATA program syntax
  - can use implicit pass-through

23

Introduction to PROC HPDS2

PROC HPDS2 differs from PROC DS2.

- All PROC HPDS2 programs execute as threads.
  - The THREAD statement is invalid
- PROC HPDS2 uses DATA= and OUT= options.
  - The DATA statement must reference **ds2gtf.out**.
  - The SET statement must reference **ds2gtf.in**.
  - Implicit pass-through is based on the IN= data set libref.

24

Introduction to PROC HPDS2

Modify a DS2 DATA program for PROC HPDS2.

```
proc ds2;
data work.new(overwrite=yes);
   dcl double Total;
   method run();
     set orion.employee_donations;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

```
proc hpds2 in=orion.employee_donations out=work.new ;
data ds2gtf.out(overwrite=yes);
 dcl double Total;
   method run();
     set ds2gtf.in;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

The IN= and OUT= data set names are PROC HPDS2 options.

26 ds06d03

Introduction to PROC HPDS2

Modify a DS2 DATA program for PROC HPDS2.

```
proc ds2;
data work.new(overwrite=yes);
   dcl double Total;
   method run();
     set orion.employee_donations;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

```
proc hpds2 in=orion.employee_donations out=work.new ;
data ds2gtf.out(overwrite=yes);
 dcl double Total;
   method run();
     set ds2gtf.in;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

Replace the data set names in the DS2 code with HPDS2 keywords.

27 ds06d03

Introduction to PROC HPDS2

Modify a DS2 DATA program for PROC HPDS2.

```
proc ds2;
data work.new(overwrite=yes);
   dcl double Total;
   method run();
     set orion.employee_donations;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

```
proc hpds2 in=orion.employee_donations out=work.new ;
data ds2gtf.out(overwrite=yes);
 dcl double Total;
   method run();
     set ds2gtf.in;
     Total=sum(Qtr1,Qtr2,Qtr3,Qtr4);
   end;
enddata;
run;
quit;
```

To execute in a High-Performance Analytics grid, use a High-Performance Analytics grid libref.

28

ds06d03

Running DS2 Code in PROC HPDS2

This demonstration illustrates executing PROC HPDS2 code in SAS.

1. Execute program **ds06d03.sas** and review the SAS log to ensure there are no errors or warnings.
2. Compare the report generated from the data set that is produced by the DS2 DATA program to that generated from the data set produced by the PROC HPDS2 DATA program. Verify that the results are identical.

Exercises

Level 1

3. Using DS2 Code in PROC HPDS2

a. Open the **ds06e03** program.

b. Convert the PROC DS2 step into a PROC HPDS2 step as follows:

1) In the PROC statement, do the following:

a) Change DS2 to **HPDS2**.

b) Add the IN= option for the **orion.banks** data set.

c) Add the OUT= option for the **work.interest** data set.

2) Change the name of the data set in the DATA statement from **work.interest** to **ds2gtf.out**.

3) Change the name of the data set in the SET statement from **orion.banks** to **ds2gtf.in**.

c. Execute the program.

1) Review the log to ensure that no warnings or errors are generated.

2) Review the PROC PRINT output to ensure that the results are as expected.

$5000 compounded quarterly

year	NAME	RATE	interest	amount
1	Carolina Bank and Trust	0.0318	$40.71	$5,160.91
5	Carolina Bank and Trust	0.0318	$46.20	$5,858.01
1	State Savings Bank	0.0321	$41.10	$5,162.44
5	State Savings Bank	0.0321	$46.71	$5,866.73
1	National Savings and Trust	0.0328	$42.02	$5,166.03
5	National Savings and Trust	0.0328	$47.88	$5,887.13

Level 2

4. Converting the PROC DS2 Step into a PROC HPDS2 Step

a. Open the **ds06e04** program.

b. PROC HPDS2 does not accept an SQL query in the SET statement. Using the query from the DS2 DATA program SET statement, add a PROC FEDSQL step before the PROC DS2 step

that creates a temporary data set named **work.ds06ex02**. This data set is used as input in the subsequent DS2 DATA program.

c. Convert the PROC DS2 step into a PROC HPDS2 step.

In the PROC statement:

1) Change DS2 to HPDS2.
2) Add the **in= work.ds06ex02** and **out= work.marginal** options.

   In the HPDS2 DATA program, replace dataset names with **ds2gtf.in** and **ds2gtf.out** references.

d. Execute the program.

1) Review the log to ensure that no warnings or errors are generated.
2) Review the PROC PRINT output to ensure that the results are as expected.

Watch List Items

SUPPLIER_NAME	PRODUCT_NAME	Product ID	Units Sold	Target	Gross Margin %	Gross Profit (Total)
Magnifico Sports	Rollerskate Roller Skates Gretzky Mvp S.B.S	240100400100	1	1.62737	25%	$38.00
Outback Outfitters Ltd	Comfort Shelter	230100700002	1	1.03529	39%	$85.00

6.3 Solutions

Solutions to Exercises

1. Executing DS2 Code in-database

a. Open the **ds06e01** program. Execute the program and review the SAS log.

Why didn't the THREAD program execute in-database?

The DS2ACCEL option was set to NO.

b. Add the DS2ACCEL=YES option to the PROC DS2 statement, and execute the program once again.

1) In the PROC statement, do the following:

a) Did the THREAD program execute in-database? Yes

b) What changes would be required to make the DATA program run in-database?

The DATA program would need to write to a data set in the ORION_DB library.

Level 2

2. Executing DS2 Code in-database.

```
/*ds06s02*/
/* After the Scoring Package code, add the following:*/
/*Create a thread to score my_db.campaign using the packaged
score_me method*/
proc ds2;
THREAD my_db.ds06s02_th/overwrite=yes;
   dcl double FinalScore;
   dcl package my_db.ds06s02_pkg m();
   method run();
      set my_db.campaign;
       /* Call the user-defined method to score the data */
      m.Score(FinalScore);
   end;
endthread;
run;
quit;

/*Create a DATA program to run the thread and DATA program in-
database*/
proc ds2 ds2accel=yes;
data my_db.ds06s02_scored/overwrite=yes;
   dcl thread my_db.ds06s02_th s();
   method run();
      set from s;
   end;
enddata;
```

```
run;
quit;
```

3. Using DS2 Code in PROC HPDS2

```
proc hpds2 in=orion.banks
           out=work.interest;
data ds2gtf.out(overwrite=yes);
   dcl double year interest amount;
   method run();
      dcl double quarter;
      set ds2gtf.in;
      Amount=5000;
      do Year=1 to 5;
         do Quarter=1 to 4;
            Interest=Amount*(rate/4);
            Amount=sum(amount,interest);
         end;
         if year in (1,5) then output;
      end;
   end;
enddata;
run;
quit;

title "$5000 compounded quarterly";
proc print data=interest;
   var Name Rate Interest Amount;
   ID year;
   format interest amount dollar10.2;
run;
title;
```

4. Converting the PROC DS2 Step into a PROC HPDS2 Step

```
proc fedsql;
create table work.ds06ex02 as
   select Supplier_Name
                ,o.Product_ID
                ,Product_Name
                ,Sum(Total_Retail_Price) as GrossSales
                ,Sum(CostPrice_Per_Unit*Quantity) as CostOfGoods
                ,Sum(Quantity) as Units
         from orion.order_fact as o
            , orion.product_dim as p
         where o.Product_ID=p.Product_ID
            and year(order_date)=2010
         group by 1,2,3
         order by 1,2
   ;
quit;

proc hpds2 in=work.ds06ex02 out=work.marginal;
```

```
data ds2gtf.out (overwrite=yes);
   dcl double Product_ID having format 12. label 'Product ID';
   dcl double GrossMargin having format percent6.2
              label 'Gross Margin %';
   dcl double Score having format 5.2 ;
   dcl double GrossProfit having format dollar10.2
              label 'Gross Profit (Total)';
   drop GrossSales CostOfGoods;
   dcl double Target;
   method ourGMP(double GS, double Cog) returns double;
      return (GS-CoG)/GS;
   end;
   method ourTgt(double U, double GM) returns double;
      dcl double Tgt;
      Tgt=.4/gm;
      return(Tgt);
   end;
   method run();
      set ds2gtf.in;
      GrossMargin=ourGMP(GrossSales,CostOfGoods);
      Target=ourTgt(Units,GrossMargin);
      If Target >= Units;
      GrossProfit=GrossSales-CostOfGoods;
   end;
enddata;
run;
quit;

title "Watch List Items";
proc print data=work.marginal noobs label;
   var SUPPLIER_NAME PRODUCT_NAME Product_ID UNITS Target
       GrossMargin GrossProfit;
   label Units ='Units Sold';
run;
title;
```

Solutions to Student Activities (Polls/Quizzes)

6.01 Multiple Choice Poll – Correct Answer

Which of these is ***not*** required to execute a DS2 thread program in an RDBMS?

a. Base SAS
b. SAS In-Database Code Accelerator for the RDBMS
c. SAS Embedded Process components installed on an RDBMS system
d. SAS High-Performance Analytics Server
e. SAS/ACCESS for the RDBMS

SAS High-Performance Analytics Server is not required to execute DS2 in an RDBMS. However, DS2 code can be executed on the SAS High-Performance Analytics Server using PROC HPDS2.

19

Chapter 7 Learning More

7.1 Learning More

Objectives

- Identify the areas of support that SAS offers.
- Identify the next steps after completing this course.

2

Next Steps

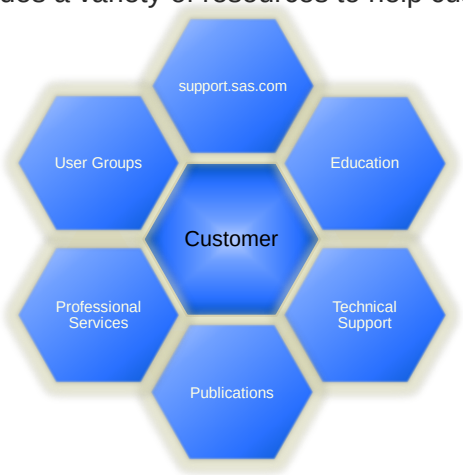
Self-Study Resources

- *SAS® 9.4 DS2 Language Reference* (**http://support.sas.com/documentation/cdl/en/ds2ref/66664/HTML/default/viewer.htm#titlepage.htm**)
- *SAS® 9.4 FedSQL Language Reference* (**http://support.sas.com/documentation/cdl/en/fedsqlref/66010/HTML/default/viewer.htm#titlepage.htm**)
- SAS Global Forum
  - 399-2013: *Leveraging Big Data Using SAS® High-Performance Analytics Server* (**http://support.sas.com/resources/papers/proceedings13/399-2013.pdf**)

3

Customer Support

SAS provides a variety of resources to help customers.



http://support.sas.com/resourcekit/

4

Education

SAS Education provides comprehensive training, including

- more than 200 course offerings
- world-class instructors
- multiple delivery methods
- worldwide training centers.

http://support.sas.com/training/

5

SAS Global Certification Program

SAS Education also provides

- globally recognized certifications
- preparation materials
- practice exams.

http://support.sas.com/certify/

6

Networking

Social media channels and user group organizations enable you to

- interact with other SAS users and SAS staff
- learn new programming tips and tricks
- get exclusive discounts.

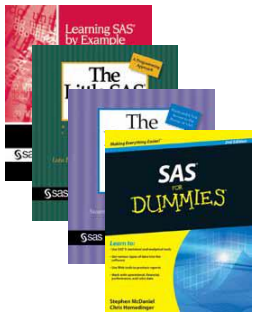
For training-specific information:

http://support.sas.com/training/socialmedia

7

SAS Books

SAS Books offers a complete selection of publications, including



- eBooks
- CD-ROM
- hard-copy books
- books written by outside authors.

http://support.sas.com/publishing/

1-800-727-3228

8

Beyond This Course

To grow your SAS skills, remember to activate the ***extended learning page*** for this course.

Free SAS University Edition software is available for you to continue practicing your new programming skills.

http://go.sas.com/free_sas

9

Next Steps

To learn more about this:	Enroll in this course:
Fully leveraging the SAS/Access Interface to Teradata	Introduction to SAS/ACCESS Interface to Teradata
Manipulating Hadoop and LASR Analytic Server data with SAS	Introduction to SAS® and Hadoop
Accessing and analyzing SAS LASR Analytic Server data	Getting Started with SAS In-Memory Statistics
Predictive modeling and machine learning with the IMSTAT procedure	Predictive Modeling Using SAS® In-Memory Statistics

10

Self-Study Resources

Online Documentation

- *SAS® 9.4 DS2 Language Reference*
- *SAS® 9.4 FedSQL Language Reference*
- *SAS® 9.4 In-Database Products: User's Guide*
- *SAS® 9.4 Language Reference: Concepts – Support for Parallel Processing*

11